Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

ingeniería en tecnología de software

.

Junio 2024

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

ingeniería en tecnología de software

.

Junio 2024

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

ingeniería en tecnología de software

.

Junio 2024

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

ingeniería en tecnología de software

.

Junio 2024

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Los miembros del Comité de Tesis recomendamos que la Tesis "A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions", realizada por el alumno Eduardo Salazar Treviño, con número de matrícula 1847972, sea aceptada para su defensa como requisito para obtener el grado de Ingeniería en Tecnología de Software.

El Comité de Tesis

Dr. Roger Z. Ríos Mercado
Director

Dra. María Angélica Salazar Aguilar
Revisor

Dr. Vincent André Lionel Boyer
Revisor

Vo. Bo.

Dr. Fernando Banda Muñoz
Subdirector Académico

San Nicolás de los Garza, Nuevo León, Junio 2024

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Los miembros del Comité de Tesis recomendamos que la Tesis "A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions", realizada por el alumno Eduardo Salazar Treviño, con número de matrícula 1847972, sea aceptada para su defensa como requisito para obtener el grado de Ingeniería en Tecnología de Software.

El Comité de Tesis

[illegible]

Dr. Roger Z. Ríos Mercado
Director

[illegible]

Dra. María Angélica Salazar Aguilar
Revisor

[illegible]

Dr. Vincent André Lionel Boyer
Revisor

Vo. Bo.

[illegible]

Dr. Fernando Banda Muñoz
Subdirector Académico

San Nicolás de los Garza, Nuevo León, Junio 2024

A los gigantes

en cuyos hombros estoy parado.

Contents

List of Figures

List of Tables

Agradecimientos

Agradezco a mis señores padres, Eduardo Salazar Naranjo y Laura Marcela Treviño Lara, por haberme apoyado en cumplir todos mis sueños y metas. Ahora que culmino mi primera etapa de estudios profesionales quiero agradecerles todo lo que han hecho por mí hasta llegar a este punto, se los debo todo a ustedes, las palabras no son suficientes para darles las gracias por su sacrificio y esfuerzo durante toda mi vida, que me han permitido soñar y cumplir esos sueños. Agradezco también a mis hermanas, Carla Elizabeth y María José por su apoyo moral y amistad fraternal, a lo largo de toda mi vida y en especial en mi carrera.

Quiero agradecer a mi principal asesor, director de tesis y mentor, el Doctor Roger Z. Ríos Mercado por toda su guía brindada en este proceso. Ha sido un camino muy largo el que he recorrido siendo impulsado y guiado por el. No existen palabras para expresar la gratitud que siento con y hacia el por su infinita paciencia, sabiduría, comentarios y retroalimentación de estos dos años. Nada de esto sería posible sin él. Adicionalmente quiero agradecer profundamente también a la Doctora Diana Lucía Huerta Muñoz, también mi coasesora por todos sus comentarios y ánimos en el proceso, si algún día sentía que iban mal las cosas, la doctora. siempre me podía animar con su optimismo, sus comentarios y atención al detalle han sido invaluables en este proceso.

Gracias a mis revisores, la doctora María Angélica Salazar Aguilar y al doctor Vincent André Lionel Boyer por su tiempo y esfuerzo en la revisión y mejora de este trabajo.

Agradezco a la Facultad de Ingeniería Mecánica y Eléctrica el haberme permitido ser alumno de la mejor institución ingenieril de México, a la Universidad Autónoma de Nuevo León por acogerme como alumno desde el bachillerato y brindarme las herramientas y oportunidades que hoy estoy aprovechando. También agradezco al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) por su apoyo económico con la beca de Ayudante de Investigador del SNI nivel III y a la clase trabajadora de México, con cuyos impuestos recaudados se financían estos programas.

Asimismo a pesar de no formar parte oficialmente del Programa del Posgrado en Ingeniería de Sistemas (PISIS) en la FIME, quiero agradecer también a sus maestros y alumnos por también haber colaborado en este proceso, la doctora Elisa, la doctora Sara Elena, el maestro Said y el doctor Sergio.

Por último, gracias a todos los amigos que he hecho en el camino: Neto, Pablo, Chava, Saúl, Emmanuel, Isaac, Uriel, Betty, Tali y Rodolfo.

Resumen

Eduardo Salazar Treviño.

Candidato para obtener el grado de Ingeniería en Tecnología de Software.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Número de páginas: 135.

Objetivos y método de estudio: Analizar, diseñar e implementar algoritmos heurísticos eficientes que proporcionen soluciones de buena calidad para un problema de optimización combinatoria de diseño territorial de una institución microfinanciera, así como comparar los resultados y tiempos de ejecución de los algoritmos propuestos con respecto a los resultados obtenidos por software de optimización exacta.

El problema de diseño territorial se refiere a la partición de un conjunto B de unidades geográficas básicas y un conjunto S de posibles centros territoriales en p territorios, respetando restricciones espaciales y de planificación.

Contribuciones y conclusiones: En esta tesis se ha desarrollado e implementado una metaheurística basada en un Procedimiento de Búsqueda Adaptativa Ávida Aleatorizada (*GRASP* en inglés) para abordar el problema de estudio, demostrando eficacia al obtener soluciones comparables a las mejores soluciones encontradas por software de optimización exacta y con mejor tiempo de cómputo.

Firma del director: ____________________________
Dr. Roger Z. Ríos Mercado

Abstract

Eduardo Salazar Treviño.

As a candidate to obtain a degree in Software Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of the study: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Number of pages: 135.

Objectives and Study Methods: The purpose of this thesis is to analyze, design, and implement efficient heuristic algorithms that provide good quality solutions for a combinatorial optimization problem arising in the territorial design for a microfinance institution.

The territorial design problem refers to the partitioning of a set B of basic geographical units and a set S of possible territorial centers into p territories. This is done while meeting spatial and planning constraints.

Contributions and Conclusions: A Greedy Randomized Adaptive Search Procedure (GRASP) has been developed and implemented to address the problem under study, demonstrating efficiency in obtaining solutions comparable to the best solutions found by exact optimization algorithms, with shorter running times.

Director signature: ______________________
Dr. Roger Z. Ríos Mercado

Resumen

Eduardo Salazar Treviño.

Candidato para obtener el grado de Ingeniería en Tecnología de Software.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Número de páginas: 114.

Objetivos y método de estudio: Analizar, diseñar, e implementar algoritmos heurísticos eficientes que proporcionen soluciones de buena calidad para un problema de optimización combinatoria de diseño territorial de una institución microfinanciera, así como comparar los resultados y tiempos de ejecución de los algoritmos propuestos con respecto a los resultados obtenidos por software de optimización exacta.

El problema de diseño territorial se refiere a la partición de un conjunto B de unidades geográficas básicas y un conjunto S de posibles centros territoriales en p territorios, respetando restricciones espaciales y de planificación.

Contribuciones y conclusiones: En esta tesis se ha desarrollado e implementado una metaheurística basada en un Procedimiento de Búsqueda Adaptativa Ávida Aleatorizada (*GRASP* en inglés) para abordar el problema de estudio, demostrando eficacia al obtener soluciones comparables a las mejores soluciones encontradas por software de optimización exacta y con mejor tiempo de cómputo.

Firma del director: [illegible]
Dr. Roger Z. Ríos Mercado

Abstract

Eduardo Salazar Treviño.

As a candidate to obtain a degree in Software Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of the study: A GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE FOR A TERRITORY DESIGN PROBLEM ARISING IN MICROFINANCIAL INSTITUTIONS.

Number of pages: 114.

OBJECTIVES AND STUDY METHODS: The purpose of this thesis is to analyze, design, and implement efficient heuristic algorithms that provide good quality solutions for a combinatorial optimization problem arising in the territorial design for a microfinance institution, as well as comparing the results and running times of the proposed algorithms with respect to the results obtained by exact optimization software.

The territorial design problem refers to the partitioning of a set B of basic geographical units and a set S of possible territorial centers into p territories. This is done while meeting spatial and planning constraints.

CONTRIBUTIONS AND CONCLUSIONS: A Greedy Randomized Greedy Adaptive Search Procedure (*GRASP*) has been developed and implemented to address the problem under study, demonstrating efficiency in obtaining solutions comparable to the best solutions found by exact optimization software, with shorter run-times.

Director signature: [illegible]
Dr. Roger Z. Ríos Mercado

Chapter 1

Introduction

A classic Territorial Design Problem (TDP), also known as a Districting Problem, organizes and partitions a given set B of small geographical units or *Basic Units* (BUs), and a set S of territory centers into p larger geographical clusters called territories or districts according to spatial constraints such as compactness and contiguity, and planning constraints in order to balance the magnitude of the performance of several activities across all territories, such as their economic performance (sales, total workload, number of customers served) or specific physical needs of each center (Ríos-Mercado and López-Pérez, 2013).

A district is called *contiguous* if it is possible to travel between each pair of basic units of the district without leaving the district. *Compactness* refers to round-shaped, undistorted territories without holes. The reason for preferring compact districts is the need of maintaining contiguity, as it minimizes daily travel distances within the territories. In terms of compactness, a dispersion measure based on the objective function of several classic location problems, such as the p-center problem or the p-median problem, is usually considered (Laporte, Nickel, and Gama, 2019). Since p is not equal to $|S|$, the first decision to be taken is which of all the S possible centers must be used to serve the BUs. Once the p centers are known, the individual allocation of BUs to their center can be decided.

1.1 Motivation

The purpose of this study is to provide a generalization for microfinancial institutions that wish to design territories representing where to open a service branch and which clients will be served by that branch. Risk balancing among the branches is extremely important for this type of business, as a failure in a single branch node can compromise the entire network.

The model described in this thesis is motivated by a previous model described in López, Ekin, Mediavilla, and Jimenez (2020). In the present model, risk balancing is modeled as a constraint, whereas in the original model it is a part of the objective function. Moreover, the model from López et al. (2020) was applied to an existing territorial design, so the objective function also included a term related to retaining as much as possible the same territorial features of the original design. Additionally, in our model we assume we do not have connectivity constraints. Finally, the previous model is solved using its mathematical formulation as a Mixed Integer Linear Programming problem.

The proposed model can be viewed as a p-median problem with multiple capacity constraints. Given that even the uncapacitated p-median problem is $\mathcal{NP}$-hard (Ríos-Mercado and Escalante, 2016), our TDP is also $\mathcal{NP}$-hard. The $\mathcal{NP}$-hard nature of the problem is what motivates us to pursue the use of heuristic approaches for solving it in a reasonable time.

1.2 Objectives

- Studying and analyzing a generalization of a territorial design problem with activity measures, risk, and center types balancing constraints.
- Designing and implementing efficient and robust heuristic and metaheuristic

algorithms for the problem.

- Demonstrating the effectiveness of solving such a problem using the proposed heuristic framework.
- Comparing the performance of the proposed robust metaheuristic with respect to a state-of-the-art exact solver.

1.3 Contribution

In this work, a Greedy Randomized Adaptive Search Procedure metaheuristic is proposed to efficiently solve a territorial design problem of a microfinancial institution.

The key components of this metaheuristic are: a robust constructive heuristic which guarantees near-feasible solutions, repaired later by a local search phase. Once a solution is feasible, the local search changes its priority towards improving the objective function using several local neighborhood structures considering a cyclical neighborhood exploration strategy. The proposed metaheuristic framework which involves parallel computing has been assessed and evaluated over a wide range of test instances.

By comparing the results of the proposed metaheuristic versus the solutions obtained from the mathematical model solved by a general-purpose optimization solver, we observe that the heuristic outperforms off-the-shelf optimizers in terms of total computing time and solution quality is within an acceptable relative optimality gap to the best bound found by the solver.

This thesis is organized as follows: In Chapter 2, we provide the literature review related to the problem. In Chapter 3, the definition of the problem and the mathematical formulation are presented. In Chapter 4, the proposed algorithms are described. Chapter 5 presents the computational experience where the proposed

metaheuristic is fully assessed. Finally, in Chapter 6, we provide the conclusions for this research work.

Chapter 2

Literature Review

In this chapter, specifically in Section 2.1, we analyze the literature for the Territorial Design Problem (TDP) and an overview on microfinancial institutions and their importance. Sections 2.2 and 2.3 describe the problem and provide the representation of feasible solutions. In Section 2.4, we will model the problem based on the previous description.

This TDP identifies potential branch offices (territory centers) and allocates clients to their respective office, balancing several constraints while minimizing the distance of the clients and their assigned institution branch.

The territory centers must be distributed across different hosts' business lines. Due to the *micro* nature of the institution, there are not enough resources to build new physical offices to give service, instead, those offices must be opened in existing facilities of different businesses and services, such as restaurants, gas stations, pharmacies, etcetera, to serve clients alongside the main activity performed by the establishment. Therefore, all centers must be distributed evenly across the different hosts' business lines.

The aim is to determine what centers to open from a set of possible locations, such that the sum of distances from the basic units to their assigned institution branch is minimized, subject to balance criteria: the required workload for each

BU, total quantity of money involved in the loans for each BU, and total utilities generated for each BU. These criteria must be balanced around a target value, and since the exact value will not be met perfectly, a tolerance parameter is introduced to allow some leeway. Finally, the sum of the risk involved in each loan for every center must not exceed a given threshold.

2.1 Districting

Districting problems (DP) or territorial design problems arise in many different contexts; from classical areas such as political districting, sales alignment, or public service, to more recent applications in police districting, waste management, commercial or healthcare (Ríos-Mercado, 2020).

The challenge of establishing political boundaries can be perceived as partitioning an administrative region, like a city or state, into subdivisions for electing political representatives, often known as *political districting.* This issue holds significant importance in democratic systems where each district selects representatives for a legislative body. To avoid politicization of the redistricting process, numerous states have established legislative and practical standards. Several criteria are used to make the districting decisions: demographic criteria such as population, minority representation and voter equality; geographic criteria that model the previously discussed compactness and contiguity features and, finally, political data.

The crucial but arduous work of developing sales and services territories is shared by all businesses that employ a sales team and require the market area to be divided into zones of accountability. Similarly, the issue of defining service territories for customer or technical facility support is closely related. In these cases, similar standards are typically used to create regions for personnel of service. There are multiple reasons for adjusting current territories, or for planning new territories. First, any increase or decrease in the number of sales or service personnel undoubtedly

requires adjustments to the territories. Additional justifications include enhancing the current workforce's reach or to distribute workload uniformly among them. Furthermore, shifts in customer demographics or the roll-out of new products require a restructuring of boundaries.

The sales territorial design problems are closely related to the problem addressed in this work. Several criteria are overlapped between both of them, for instance, the number of territories is predetermined and the presence of basic units, that represent individual customers assigned to larger geographical units with exclusive assignments are considered. Sales districting problems also have activity-related criteria that represent the performance of the products or service that an organization may offer. Dealing with sales and service territory districting issues, there is often a pursuit for districts that balance one or more characteristics (referred to as *activity measures*). This standard shows the relationship between territories to ensure equitable treatment of all sales personnel (Kalcsics, 2005).

The creation of service districts emerges in multiple scenarios. One of them is focused on social infrastructures: hospitals or public utilities. There are cases where districts are required to assign each resident to a specific facility for service provision, such as routine health check-ups, or to delimit areas of responsibility for home-care visits by healthcare staff such as nurses or physiotherapists. The objective is to identify joint districts that are easily reachable via public transport and ensure a balanced workload based on service and travel time or aim for optimal capacity usage of the social facility.

Despite all the different criteria of all the possible applications of districting, there is no definitive mathematical model to represent each problem. Depending on each context, the formulation of the problem and the solution approach are different. The compactness criterion is also something that depends on the specific problem as there is no concrete definition of compactness since it depends on the representation of the basic units. In political districting, compactness is preferred to be represented

as specific territorial shapes, while in sales and services districting, distance-based compactness is preferred.

For comprehensive surveys and discussion on districting models, algorithms, and applications, the reader is referred to the works by Kalcsics and Ríos-Mercado (2019) and Ríos-Mercado (2020).

The studied TDP considers a distance-based dispersion measure, which can be represented in several manners, resulting in several models and solutions. The commonly used dispersion measures are those used for the p-Median Problem (pMP) and the p-Center Problem (pCP). In the pMP, the objective is to place p facilities (or medians) in such a way that the total weighted distance (or cost) between demand points and their assigned facilities is minimized (Laporte, Nickel, and Gama, 2019). This is particularly suitable for situations where the total travel (or service) cost needs to be minimized, for example: to determine the optimal locations for warehouses to minimize the total transportation cost. In contrast, the pCP aims to locate p facilities (or centers) such that the maximum distance from a demand point to the nearest facility is minimized (Medrano, 2020). This essentially minimizes the worst-case scenario in terms of the distance traveled by a customer to reach a facility. This might be used, for example, to locate emergency facilities such as fire stations or hospitals, where the aim is to minimize the longest distance that must be traveled in an emergency. In the context of districting, Salazar-Aguilar et al. (2011) present a computational study of both pMP and pCP objectives used within a territory design framework. For this thesis, the dispersion is measured using a function from the p-Median problem.

2.2 Microfinancial institutions

Microfinancial institutions are important to reduce income inequality and poverty (Khandker, 2005). They offer credit and other financial products to sectors of the

population that are normally rejected by commercial banks as they represent high-risk with low profit options. They have risen as an alternative to regular banking, especially in countries with developing economies (Bruton, Khavul, Siegel, and Wright, 2015). Due to their fragile nature, the risk associated with the loans granted must be balanced across the branch offices.

Compared to traditional banking systems, the MFIs do not operate nor own branch offices, instead, only overseeing credit operations. These branch offices or subsidiaries operate as part of local businesses (gas stations or grocery stores) that allow MFIs to reduce operating and construction costs. MFIs face risks that normal banks do not. Traditional banking systems discourage the practice of lending money to people who do not have guarantees to secure the loans and are unable to pay them back, excluding vulnerable sectors of the population.

A relationship of mutual rewards with the branches is established so that the network remains sustainable in the long term. Balancing risks between different branches is important, since there is no universal definition of risk, measures can be based on statistical deviations or problem-specific functions.

In this thesis, a function based on the problem's variance is used, which will be referred to as the variance of excessive gain. For each BU, a gain variance corresponds to a higher risk in which small establishments are not exposed to high variance. Furthermore, each open branch office and their corresponding served clients, forms a territory or district, with the office being the territory center and the clients the basic units.

Also, each type of branch must manage different levels of risk in order to control and balance the tolerance of each territorial center. For this, the gain variance for each BU is first estimated using historical customer data. Then, the total gain variance within the centers is used as an internal risk measure to obtain information about the customer retention rate. This rate estimates how probable it is that a client continues to be such. Therefore, the MFI pre-determines an upper limit for

the total weighted gain variance, for each territorial center. Subsequently, customers should be assigned to territories in such a way that the total variance of excessive gain is minimized. Deviations towards lower limits are not taken into account. A more detailed description of the studied problem in this thesis is provided below.

2.3 Districting applied to MFIs

Territorial design models are rarely applied to traditional banking in the literature. The most important contribution to the literature can be found in Monteiro (2005) and Monteiro and Fontes (2006), in which traditional MILP formulations are applied to locating and maintaining bank-branches, closing or opening existing branches by posing a coverage problem. However, these contributions recognize the shortcomings of the MILP approach and introduce local search heuristics that start from a partial solution provided by an optimization solver. These contributions take into consideration several costs of operation, location, as well as sizing, thus they are not exactly districting formulations.

Districting, specifically applied to MFIs, first appeared in López, Ekin, Mediavilla, and Jimenez (2015) where a two phased mixed integer program, hybridized with several heuristics, is used to solve a real-life instance from a MFI located in Monterrey, México. This model presents the risk balancing as a parameter in the objective function as well as introducing a similarity parameter, which motivates the new design to retain the features of a previous design. Additionally, another constraint limits the maximum amount of distance traveled by the customers. Some BUs are constrained to be allocated to different territories, introducing disjoint assignment constraints. Finally, contiguity constraints, which evaluate that every district induces a connected graph between all the BUs and the corresponding territory center, are included. This type of constraint in specific grows exponentially, so solving medium to large instances becomes intractable in a reasonable computing time. By using heuristics in several phases, they were able to reduce their problem complexity

in order to use an exact solver.

A follow-up work was published in López, Ekin, Mediavilla, and Jimenez (2020), in which the heuristics used to set some variables for the solver were slightly improved on, using Geographic Information System (GIS) software to provide interpretability to the MFI and properly implement the solution found. As it can be seen, there have been no previous attempts to apply districting to MFIs in a broader context, not tied to a specific case study.

In this thesis, we propose a generalization of the model of López, Ekin, Mediavilla, and Jimenez (2020) so it can be applied in more cases by removing: the hard constraints on maximum distance, the disjoint assignments constraints, and the contiguity constraints, as well as simplifying the objective function by considering the risk term as a constraint and removing the similarity to a previous plan parameter. Notice that most territorial design problems have a district center c_k for each district D_k that coincides with a basic unit, i.e. $c_k \in B$: however, in this thesis, the district center is a member of another set, $c_k \in S$. By reformulating the problem and solving it with an exact solver, we can verify that the computation is not trivial and an optimal solution cannot be reached in reasonable time; thus, we encourage the use of heuristics and metaheuristics to solve the problem. This thesis is the first work, to the best of our knowledge, to propose a generalized heuristic framework for districting applied to MFIs and, subsequently, the first to propose metaheuristics.

Chapter 3

Problem description and Mathematical model

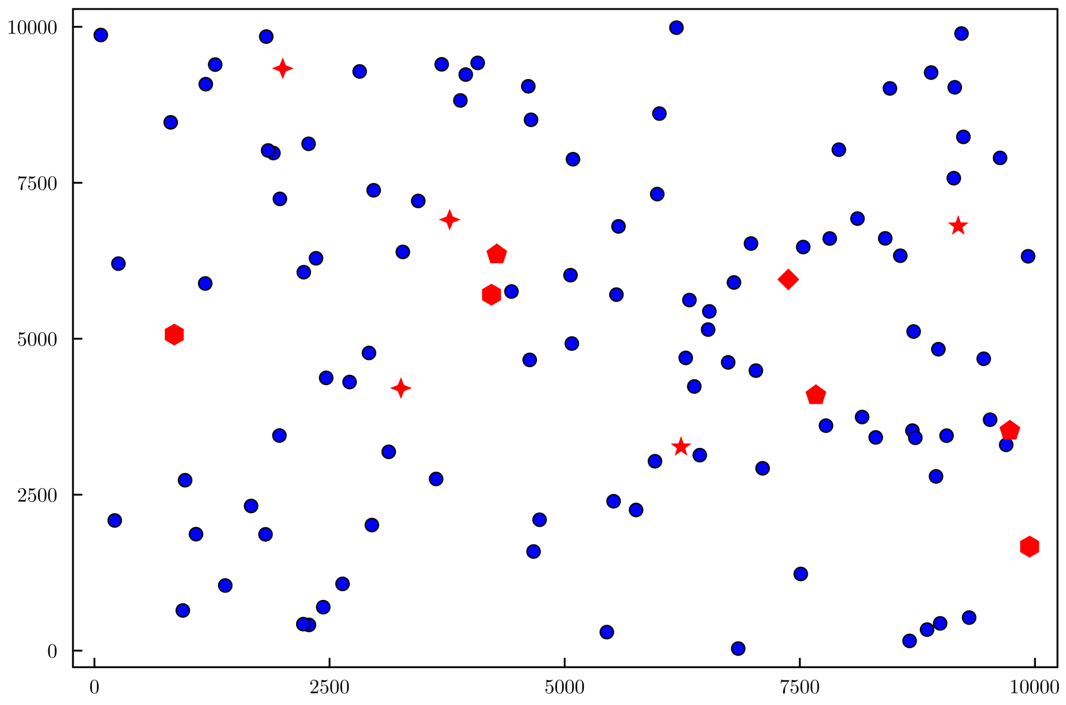
3.1 Problem description

Given a set S of possible territory centers, a set B of BUs, a distance matrix d and a p number of centers to be open, let the set M of activities to evaluate and their corresponding activity values v_m^j, where $m \in M$ and $j \in B$, the sum of these values for all the BUs assigned to a territory center must be in a range defined by multiplying the target measure μ_m^i for each activity $m \in M$, $i \in S$, and a t_m tolerance parameter. Additionally, given each individual risk value r_j for $j \in B$, the sum of all the risk values of the assigned BUs to a center i must not exceed a risk threshold β_i.

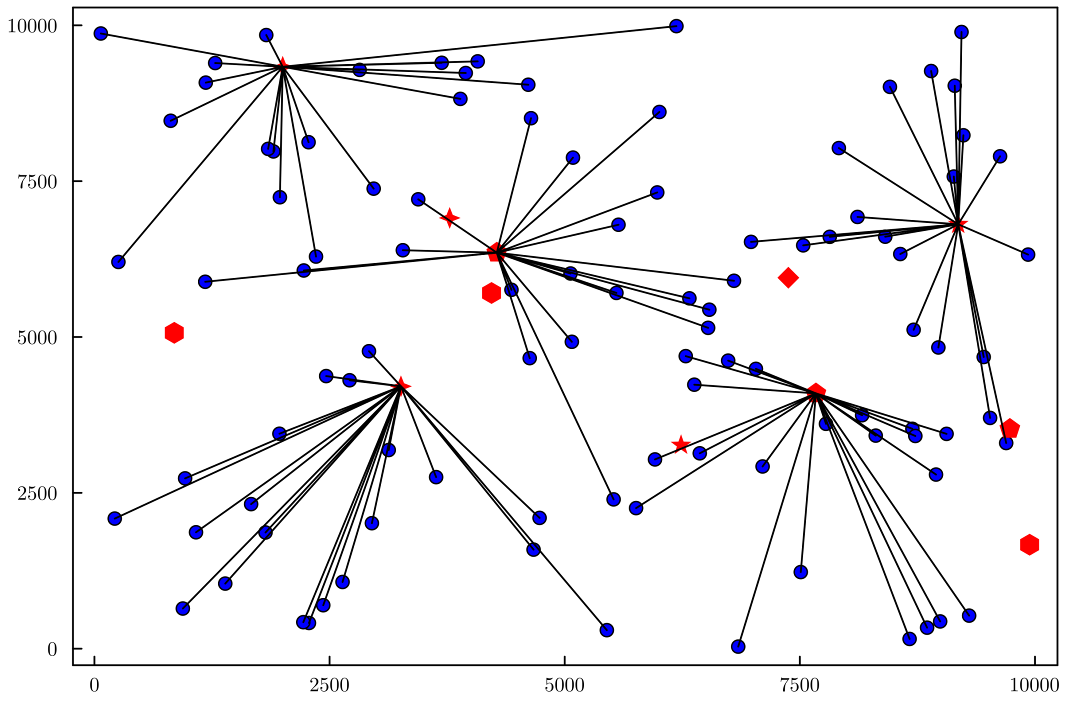
Given a set K of territory types, an associated T_{ik} value is assigned for each center $i \in S$ of type k, for $k \in K$. A lower bound L_k and an upper bound U_k, are determined for the number of centers used for each type, $\forall k \in K$. Finally, each BU $j \in B$ must be assigned only once. The territorial design problem aims to find the location of the p centers to be opened and the allocations of all BUs to their corresponding centers such that the sum of the distances in d between each BU and its assigned center is minimized, taking into account the previous constraints.

3.2 Illustrative example of feasibility

Figure 3.1a shows a basic example of an instance of the problem with BUs and territory centers, randomly located. The BUs are represented as blue circles, and the different red geometric figures are used to represent the different center types, with $K = 5$. In contrast, Figure 3.1b illustrates the optimal solution to the instance, minimizing the total distance between each BU and their assigned territory center. It is important to emphasize that there are only 5 centers available to serve BUs, as $p = 5$. In an instance of larger size, the optimal solution that minimizes the sum of distances may allocate individual BUs to centers that are not their nearest, in a counter-intuitive fashion. These allocations arise as the territories are constrained by the activity measures and the risk thresholds, as seen in Figure 3.2b.

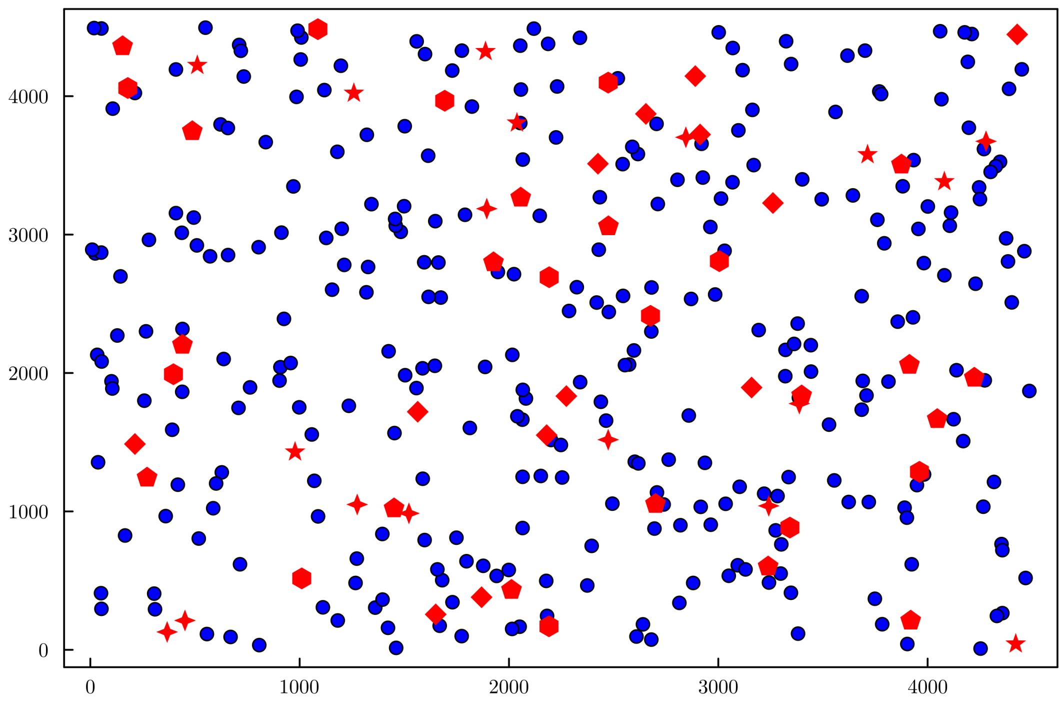


(a) Example of an instance with $|B| = 100$, $|S| = 12$, $p = 5$

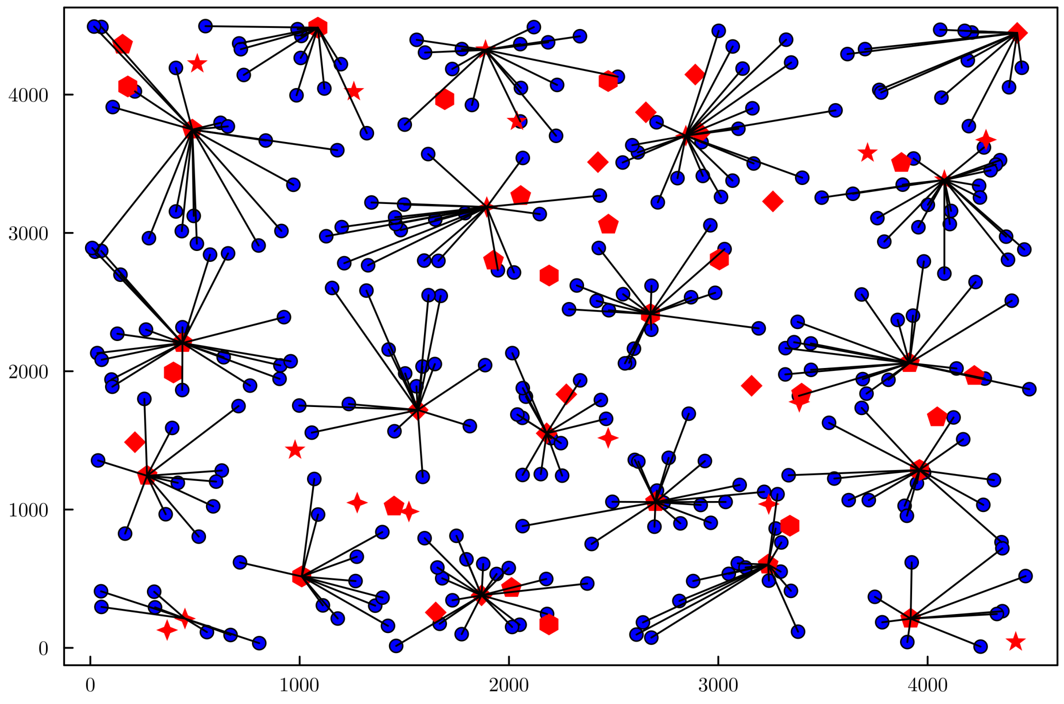


(b) The optimal solution for the instance.

Figure 3.1: Optimal solution with a weight of 208567 for a small size instance



(a) Instance with $|B| = 300$, $|S| = 60$, $p = 20$



(b) Optimal solution for the above instance.

Figure 3.2: Optimal solution for a larger instance

3.3 Mathematical model

In this section, the mathematical model used in this thesis is described, studied, and analyzed. This model is based on the formulation presented by López, Ekin, Mediavilla, and Jimenez (2020) with the following modifications. The objective function in the original model has three terms to minimize: the sum of distances, the risk of the allocations, and a similarity factor that measures how similar the new territorial design is to an existing design. As this version of the problem models the risk as a constraint and has no previous territorial design, the objective function only minimizes the distance. Additionally, the previous model has a specific connectivity constraint set that requires all territories to induce a connected graph with the BUs and their corresponding center. This set has an exponential size, thus they limit this restriction to a given set of territory centers only. On the other hand, there is also a restriction that ensures that certain pairs of BUs are allocated to different territories due to geographical or management requirements. Both of the connectivity and the exclusive assignments are not present in the current model given the additional complexity they would represent.

- Sets and parameters:

- S : Set of possible territory centers
- B : Set of possible BUs
- M: Set of activities to evaluate
- K: Set of territory center types
- d_{ij}: Distance between center i and BU j, $i \in S, j \in B$
- μ_m^i: Target of activity m measured at center i, $i \in S$, $\forall m \in M$
- v_m^j: Measure of activity m at BU j, $j \in B$, $\forall m \in M$

- t_m: Tolerance of activity m measure, $\forall m \in M$
- r_j: Risk value at BU j, $j \in B$
- β_i: Risk threshold at center i, $i \in S$
- $T_{ik} = 1$ if center i is of type k, 0 otherwise, $i \in S, k \in K$
- $L_k, U_k =$ Lower and upper bounds of number of centers of type k to be located, $k \in K$
- $p =$ Number of centers to be used

- Decision variables:

- $Y_i = 1$ if center i is open; 0 otherwise.
- $X_{ij} = 1$ if BU j is assigned to center i; 0 otherwise.

Then, the territorial design problem for the MFI is defined as:

$$\min \sum_{i \in S, j \in B} d_{ij} X_{ij} \tag{3.1}$$

Subject to

$$\sum_{i \in S} X_{ij} = 1, \qquad \forall j \in B \tag{3.2}$$

$$X_{ij} \leq Y_i, \qquad \forall i \in S, j \in B \tag{3.3}$$

$$\mu_m^i (1 - t_m) Y_i \leq \sum_{j \in B} v_m^j X_{ij} \leq \mu_m^i (1 + t_m) Y_i, \qquad \forall i \in S, \forall m \in M \tag{3.4}$$

$$L_k \leq \sum_{i \in S} T_{ik} Y_i \leq U_k, \qquad \forall k \in K \tag{3.5}$$

$$\sum_{i \in S} Y_i = p \tag{3.6}$$

$$\sum_{j \in B} r_j X_{ij} \leq \beta_i, \qquad \forall i \in S \tag{3.7}$$

$$X_{ij} \in \{0,1\}, Y_i \in \{0,1\}, \qquad \forall i \in S, \forall j \in B \tag{3.8}$$

The objective function (3.1) minimizes the sum of the distances between the BUs and their assigned center. Constraints (3.2) state that each BU j must be assigned to a single center i. Constraints (3.3) check that the BUs are only assigned to open centers. Constraints (3.4) establish that the activity m measured for each territory i, must be within a tolerance range defined by the t parameter. Notice that because of the discrete structure of the problem it is impossible to perfectly balance each territory. Constraints (3.5) limit the located centers' types within their corresponding lower and upper bound. Constraint (3.6) ensures that only p territory centers are considered. Constraints (3.7) specify that the total risk value for each territory, must not surpass its defined risk threshold. Finally, the binary nature of the variables is expressed in Constraints (3.8).

Therefore, for an instance with $|S|$ possible centers and $|B|$ basic units, the total number of decision variables is $|S||B| + |B|$.

The problem is $\mathcal{NP}$-hard. This can be argued as follows. Clearly, the problem is in $\mathcal{NP}$ because one can check feasibility of a given solution in polynomial time. Now, by reduction, if we take a special case of our problem with a very large value for parameters t_m, U_k, and β_i and a value of zero for all L_k, Constraints (3.4), (3.5), and (3.7) become redundant so we are left with a p-median problem. Thus, we have proven that the p-median problem is polynomially reducible to our problem, and since it is well-known that the p-median problem is $\mathcal{NP}$-hard, so is our problem.

Chapter 4

A Greedy Randomized Adaptive Search Procedure

In this chapter, a Greedy Randomized Adaptive Search Procedure metaheuristic is presented, which includes several well-known heuristics adapted to the problem studied in this thesis.

Although some exact optimization approaches exist for territorial design problems (Salazar-Aguilar, Ríos-Mercado, and Cabrera-Ríos (2011), Ríos-Mercado and Bard (2019), Sandoval, Díaz, and Ríos-Mercado (2020)), the vast majority of the work has been on metaheuristics, given the inherent complexity of territory design problems.

First introduced in Feo and Resende (1989) as a "probabilistic heuristic", the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic is a multi-phase iterative process designed for solving combinatorial optimization problems. It combines elements of greediness and randomness to effectively navigate the solution space. The two main components of GRASP are the construction phase and the local search phase, which work to iteratively improve upon solutions.

In the construction phase, an initial feasible solution is built, starting from an empty solution and iteratively adding elements to it. At each step, a candidate

list of possible next elements is generated based on a greedy function. The key aspect of GRASP's construction phase is the incorporation of randomness: instead of choosing the best candidate (as in a purely greedy approach), one element is randomly selected from a subset of potential candidates. This subset, often referred to as the Restricted Candidate List (RCL), is formed by selecting elements that are within a certain α quality threshold or a fixed size. In this thesis, we use the first approach. The randomness introduced at this stage helps in diversifying the search, avoiding premature convergence to local optima.

Once the construction phase generates an initial solution, the local search phase seeks to improve it through an iterative process. The solution is refined by exploring its neighborhood, which is a set of solutions that can be reached through small modifications (like swapping, adding, or removing elements). The local search continues until a local optimum is found, meaning no better solution can be found in the neighborhood. This process is repeated until a stopping criterion is met, typically, a fixed number of iterations.

Since each iteration is independent, generating a different initial solution every time the construction phase is performed, which is then improved during the local search phase, GRASP has the potential to be parallelized, running each iteration on a different computer core thread. This repeated process allows the algorithm to explore various parts of the solution space, naturally balancing diversification (exploring different regions of the solution space) and intensification (searching a promising region). The randomness in the construction phase promotes diversification, while the local search phase focuses on intensification (Feo and Resende, 1995).

GRASP has been very successfully applied to a class of districting and territory design problems (Ríos-Mercado and Fernández, 2009; Fernández et al., 2010; Ríos-Mercado and Escalante, 2016) including multi-objective optimization formulations (Salazar-Aguilar, Ríos-Mercado, and González-Velarde, 2013).

In order to build a GRASP metaheuristic framework, a constructive and local

search heuristics must be developed first, to integrate them into the framework. In this chapter we define what heuristics and strategies were explored to build the proposed GRASP. The pseudocode of the GRASP is presented in Algorithm 1.

Algorithm 1 `GRASP(Instance,` $\alpha_l, \alpha_a, i_{\max}$`)`

Input: Instance = instance of the problem; α_l = RCL threshold parameter for location phase; α_a = RCL threshold parameter for allocation phase; $i_{\max}$ = maximum number of iterations.

Output: X^* allocation matrix of BUs, Y^* location vector of centers, z^* objective function value

```
1:  $X^* \leftarrow \emptyset$
2:  $Y^* \leftarrow \emptyset$
3:  $z^* \leftarrow \infty$
4:  $i \leftarrow 0$
5:  $z \leftarrow f(X, Y)$
6:  while $i < i_{\max}$ do
7:      $Y \leftarrow$ p-disp-grasp(Instance, $\alpha_l$)
8:      $X \leftarrow$ opp-cost-queue-grasp(Instance, $Y, \alpha_a$)
9:      $(X', Y') \leftarrow$ local-search(Instance, $X$, $Y$)
10:     if $f(X', Y') < z^*$ then
11:         $X^* \leftarrow X'$
12:         $Y^* \leftarrow Y'$
13:         $z^* \leftarrow z(X', Y')$
14:     end if
15:     $i \leftarrow i + 1$
16: end while
17: return $X^*, Y^*, z^*$
```

4.1 Constructive Heuristic

The foundational mathematical programming model for districting was introduced by Hess, Weaver, Siegfeldt, Whelan, and Zitlau (1965). In this paper, the *location-allocation* heuristic was first introduced. This heuristic approach breaks down the intertwined tasks of location and allocation inherent to the districting problem into two separate stages. These two stages, one focusing on the selection of territory centers and the other on the assignment of basic units to these centers, are executed in a repetitive cycle until a satisfactory outcome is achieved. The process begins with the location phase, where the centers of future territories are identified, followed by the allocation phase, which involves the distribution of basic units to the selected centers.

The constructive heuristic presented in this thesis follows the same location allocation strategy. In the location phase, the centers, or branches of the MFI, are selected to serve the BUs, while in the allocation phase, the BUs are individually allocated to an open center. Several approaches for both phases were tested and will be presented in the following sections.

4.1.1 Location Phase

Previous studies including Ríos-Mercado, Álvarez Socarrás, Castrillón, and López-Locés (2021) use the location-allocation heuristic successfully by first solving the location phase using p-partition-based heuristics to initialize the set of centroids. Another method for obtaining this initial set of territory centers is by solving the Mixed Integer Linear Programming (MILP) formulation until a feasible solution is found, however, due to the computationally complex nature of the MILP, it does not scale well for larger instances for the problem, this is shown in the next chapter. We believe it is important to once again remind the reader that classic formulations

of the TDP typically contain a single set that includes both the possible territory centers and the basic units, making them interchangeable; however, in this thesis, there are two different sets, one for centers and another one for the BUs. This adds further complexity to our problem as the heuristics used in previous works for this phase need to be adapted to our needs, more specifically the p-dispersion problem heuristic.

Two strategies were explored to use as the location phase algorithm: a p-dispersion problem heuristic and semi-linear programming. Of these, the p-dispersion problem was chosen as the algorithm to use.

4.1.1.1 The p-Dispersion Problem

Due to p being a parameter of the problem, a common strategy is to use p-partition-based heuristics to first divide the set of centers into p subsets, as stated in the previous section. Cano-Belmán et al. (2012) use the p-dispersion problem to locate the centers to be used. The p-dispersion problem, also known as the maxmin problem, aims to select a set of p points that maximizes the minimum distance between any selected pair of points in the set (Erkut and Neuman, 1991).

We are going to use the *Greedy Constructive* (GC) heuristic proposed by Erkut et al. (1994), to try to locate p territory centers from the S set, following the intuitive idea that if they are evenly dispersed, the BUs will have a close enough center to be allocated to. The use of this specific heuristic is reinforced by the contribution of Ravi et al. (1994) that states that there is no polynomial time algorithm that can provide a solution that is not worse by a factor of 2 compared to the optimal solution (unless $\mathcal{P} = \mathcal{NP}$) to the p-dispersion problem, as well as the fact that one of the conclusions of Erkut et al. (1994) is that the selected heuristic presents good enough results for problems in which p is a small proportion of all the points in the problem, which is the case for this TDP. This heuristic is presented in Algorithm 2.

Algorithm 2 GC(D', S, p)

```
Input: $D'$ distance matrix, $S$ set of nodes, $p$ number of nodes to select out of $S$
Output: $Y$ solution set $p$ centers
1: $Y \leftarrow \emptyset$
2: Find $x_1$ and $x_2$ in $S$ such that $d(x_1, x_2) = \max\{D'(x_i, x_j) : 1 \leq i < j \leq p\}$
3: $Y \leftarrow \{x_1, x_2\}$
4: while $|Y| < p$ do
5:     Find $x_i \in S \setminus Y$ such that $d(x_i, Y) = \max\{d(x, Y) : x \in S \setminus Y\}$
6:     $Y \leftarrow Y \cup \{x_i\}$
7: end while
8: return Y
```

The heuristic works by initializing an empty solution set Y on line 1, then it selects the two furthest apart points on the set S as shown on line 2, and adds them to Y.

Then it iterates until the solution set reaches p as its size, from lines 4 to 7. During each iteration, the heuristic finds the node that maximizes the minimum distance to Y and adds it to the solution set, as shown in lines 5 and 6. The distance of an individual node to the set is defined as the minimum distance of the candidate node x_i to each member of Y. Expressed in a more formal notation, the distance between a point x_i and a set Y is defined as $d(x_i, Y) = \min\{D'(x_i, x_j) : x_j \in Y, x_i \neq x_j\}$.

However, one must remember that the territorial centers have an associated T_{ik} individual business type, therefore the types must be also taken into consideration while dispersing the centers, as there is a minimum and a maximum number of centers to be used from each type.

By solving $|K|$ p-dispersion problems, considering only centers of a given $k, k \in K$ type for each problem and using $p = L_k$, we obtain an initial set of selected centers that ensures that fulfills the minimum number allowed for each k. Then, by taking

the union of the resulting centers, the Y vector is obtained as the Algorithm 3 shows. An important thing to note is that the overall p of the problem never equals the sum of all L_k, so a second phase after the union of the k sub-problems is required in order to reach the p centers required to be opened.

This phase solves yet another p-dispersion problem, taking the union of the previous centers as a starting set, and selects the most disperse node to Y, as long as the node respects the U_k constraint for its corresponding k type, until $|Y| = p$.

Algorithm 3 `p-dispersion for Y location`$(d, p, T_{ik}, L_k, U_k, S)$

```
Input: $d$ distance matrix, $p$ centers to open, $T_{ik}$ binary matrix of center types, $L_k$, minimum number of centers to be used of type $k$, $U_k$ maximum centers to be used of type $k$, $S$ set of centers
Output: $Y$ decision vector of located centers
 1: Compute distance matrices, $\forall k \in K$
 2: for $k \in K$ do
 3:     $S_k \leftarrow \{i \in S \mid T_{ik} = 1\}$ { $S_k$ represents all centers of type $k$}
 4:     $P\star \leftarrow$ GC(distance matrix of $k$, $L_k$, $S_k$) {Solve the problem for each $k$, with its corresponding nodes and distance matrix}
 5:     $Y \leftarrow Y \cup \{P\star\}$ {Build $Y$ upon the partial solution}
 6: end for
 7: while $|Y| < p$ do
 8:     for node $\in S$ do
 9:         if node $\notin Y$ then
10:             Determine type of node in $S_k$
11:             Check if $U_k$ for this node type is reached; if so, skip this node
12:             Compute the minimum distance from node to all nodes in $Y$
13:             Store this minimum distance.
14:         end if
15:     end for
16:     Select the node with the maximum minimum distance
17:     Add this node to $Y$
18: end while
19: return $Y$
```

A graphical step-by-step example of the algorithm is shown in Section A.6.1.

4.1.1.2 Partial Linear Programming Relaxation

Since our problem is an $\mathcal{NP}$-hard problem according to Karp (1972), large instances are intractable by branch-and-bound methods. By introducing an integer relaxation in the X_{ij} decision variables and leaving the Y_i variables as integers, the idea is to provide to the optimization solver an easier problem to solve, meaning faster times to reach a feasible solution. The solver will provide us with valid results for Y_i, meaning the territory centers have been located. However, this solution may not represent a feasible solution of the original problem; as the X_{ij} variables are not required to be integer. But, we can use the solution values of Y as valid for the proposed heuristic. This approach is referred to as the Partial Linear Programming Relaxation (PLP).

Both of these approaches were used during the experimentation phase of the heuristic to determine which of these two strategies was the one going to be integrated in the GRASP metaheuristic. In the end, the p-dispersion problem heuristic was used as the first phase of the algorithm. The results behind this decision are detailed in Chapter 5.

4.1.2 Allocation Phase

Once the territory centers have been located, the allocation phase of our approach will allocate each individual BU to an open center. This phase is the core of the heuristic as the objective function of the problem directly measures how well clients are allocated to their nearest center whilst complying with the balancing constraints. A possible greedy approach would be to simply allocate all clients to their nearest center, but this provides infeasible solutions. Therefore, a more intelligent decision process must be introduced to achieve the maximum degree of feasibility as possible before passing the constructed solution to a repair phase.

For this phase, two algorithms are proposed, both based on the concept of

opportunity cost. The first algorithm is referred to as the Opportunity Cost with Matrix and the second one is the Opportunity Cost with Queue. The latter was chosen to be used as the allocation algorithm in the GRASP.

4.1.2.1 Opportunity Cost with Matrix

The opportunity cost refers to the value of the next best alternative that is foregone when a decision is made to choose one option over another. It represents the benefits that could have been gained by choosing an specific alternative option. This concept is crucial in economics and decision-making, as it helps in understanding the potential costs involved in not selecting the next best alternative.

Due to the constrained nature of the problem, which is limited in resources, the choice of where to allocate each BU must be made from several mutually exclusive centers. As BUs are assigned to a center, the balancing constraints of both the activities and risk measure will prevent the assignment of any more BUs to that specific center, introducing the scarcity in the mutually exclusive choices available.

By calculating the opportunity cost of all BUs as the difference of the minimal distance available to a center and all other centers, and by storing the results in a matrix with the same size as d, we can now use a heuristic that minimizes the opportunity costs across all BUs, using this matrix as the data structure which will guide the algorithm.

During the allocation phase, we are computing values for a specific i center and a specific j BU. For feasibility checks of the constructed solution, we can keep track and update only the values affected by these specific i and j. By using additional data structures, we introduce the concept of partial evaluation: instead of computing and checking the feasibility of the whole solution, we only update and check it in each individual allocation, making these computations more efficient.

For the algorithm, we start a while loop that iterates until every BU has been assigned. In this loop, we will find the indices of the n largest opportunity costs across the opportunity cost matrix, where n is a parameter of the algorithm. For each index j, which represents the BU, we find the nearest center i to this BU, taking into account that $Y_i = 1$. Afterwards, we compute and check for any violated constraint with the respective allocation of i and j. The number of violated constraints is stored in an array, from which we will pick the allocation that is feasible. If all possible allocations are infeasible, we perform another search across the matrix, checking for the next n nearest centers to the first BU in the list, as well as computing again the constraints, by eventually allocating to the center with the least amount of violated constraints. When the j BU is set to 1 in the X matrix, it is marked as assigned for the algorithm.

The time complexity for the worst case scenario of this allocation strategy is $\mathcal{O}(|B|^2 \cdot |S|)$ time, as there are constant operations of finding either the n maximum or minimum values across the whole matrix, of size $|S| \cdot |B|$.

Notice that the minimum and maximum distances do not really change over the algorithm, thus an initial precomputation of those indices may be useful to speed up times. With those insights, we present the second approach to allocate BUs.

4.1.2.2 Opportunity Cost enhanced with Data Structures

Instead of computing a matrix with all the possible costs, we only compute the opportunity cost from the best possible assignment and the worst one for each BU, storing that difference in a priority queue. A priority queue is a specialized data structure that is similar to a regular queue or stack, but where each element has a priority associated with it. In a priority queue, an element with higher priority is served before an element with lower priority. If two elements have the same priority, they are served according to their order in the queue. In this case, the opportunity

cost is the priority associated with each BU. By traversing this queue, we allocate the BUs in order of largest to least opportunity cost. The n best possible assignments for each BU do not change in distance, therefore we can sort them as well prior to the algorithm, storing the ordered centers in a dictionary. n was set to p to evaluate all the possible assignments.

Algorithm 4 defines the function that calculates such data structures, with *pq* being the priority queue for the opportunity cost and *best_assignments* the dictionary that stores the sorted centers nearest to each client.

Algorithm 4 `compute-structures`(d, Y, B, S, n)

```
Input: Distance matrix $d$, Decision Variable $Y$, $B$ set of BUs, $S$ set of BUs, $n$ top
    assignments
Output: Best assignments dictionary best_assignments, Priority queue pq
1: Initialize best_assignments as an empty dictionary
2: Initialize pq as a priority queue with reverse order
3: for each $j \in B$ do
4:     Initialize an array costs to store opportunity costs for $j$
5:     for $i \in Y$ do
6:         Store $d_{i,j}$ along with index $i$ in costs
7:     end for
8:     Sort costs in ascending order of cost
9:     Store the indices of the top $n$ facilities with the lowest cost in
       best_assignments[$j$]
10:    Calculate the opportunity cost for $j$ as the difference between the highest
       and lowest cost
11:    Add to the queue the $j$ BU and its opportunity cost into pq
12: end for
13: return best_assignments, pq
```

We reuse the data structures to keep track of the constraints and their status

throughout the traversal of the queue. During the queue traversal, we will mark centers as full when their respective activity measures' meet the lower bound target, forcing the algorithm to diversify and choose another center, and ensuring an even distribution among centers. Since the algorithm respects the lower bounds and it simultaneously prevents exceeding upper bounds, feasible solutions are produced more consistently than the opportunity cost with matrix algorithm in a faster time,improving the worst-case scenario from $\mathcal{O}(|B|^2 \cdot |S|)$ to $\mathcal{O}(|B| \cdot |S|)$.

However, it is still possible that a BU is left unassigned due to all of the centers being marked as full, so we must define a function to handle unassigned BUs. This is a very simple function that will simply assign those BUs to the center that has the most capacity left in the β risk threshold.

This algorithm is better in every way than the previous approach of the opportunity cost stored in a matrix, due to the use of data structures and a better understanding of the problem and exploitation of its underlying structure. A graphical step-by-step example of this algorithm is shown in Section A.6.2

4.2 GRASP Metaheuristic Adaptations

The proposed GRASP metaheuristic required an adaptation of the mentioned algorithms to introduce the concept of the Restricted Candidate List (RCL). In this case, we use a value-limited RCL using two α parameters: α_l for the location phase and α_a for the allocation phase. In Chapter 5, we present the results that guide our decision towards using the p-dispersion strategy for the location phase and the opportunity cost enhanced with the queue and other data structures as the allocation phase. In this section, we present the corresponding modifications in order to introduce the RCL in our GRASP.

Algorithm 5 implements a GRASP (Greedy Randomized Adaptive Search Procedure) heuristic for the p-dispersion problem. The procedure is as follows:

Taking as input an instance of the problem comprising a distance matrix d, a set of points S, the number of centers p to select, and a quality threshold α_l for constructing the Restricted Candidate List (RCL), the algorithm identifies the pair $(x_i, x_j) \in S$ that maximizes d_{x_i,x_j}. These points form the initial solution set $Y = x_i, x_j$.

In order to iteratively construct the solution, the algorithm iterates while $|Y| < p$, using the greedy function $\phi : S \setminus Y \to \mathbb{R}$ as $\phi(x_i) = \min\{d_{x_i,x_j} : x_j \in Y\}$, which computes the minimum distance from a candidate point to the current solution set, and determines $\phi_{\max} = \max\{\phi(x_i) : v \in S \setminus Y\}$ and $\phi_{\min} = \min\{\phi(x_i) : x_i \in S \setminus Y\}$. With these two values, it constructs the RCL as
RCL: $\{x_i \in S \setminus Y : \phi(x_i) \geq \phi_{\max} - \alpha_l(\phi_{\max} - \phi_{\min})\}$. This set comprises candidate points that are relatively distant from the current solution set, and finally it randomly selects a point $x_i \in$ RCL and updates $Y = Y \cup \{x_i\}$.

This approach employs a balance between greedy selection and randomization. The greedy component favors points that maximize the minimum distance to the current solution set, while the randomized component introduces diversity by selecting from a set of good candidates. The parameter $\alpha_l \in [0, 1]$ controls the restrictiveness of the RCL: as α_l goes to 0, the algorithm becomes more greedy, whereas larger values of α_l introduce more randomness into the selection process.

It is important to notice that this algorithm is used to solve the k sub-problems for each center type. Once the k solutions have been joined into the larger set, it reuses the same ϕ greedy function, adding centers to the RCL, as long as the candidate center respects the U_k constraint of its corresponding k type.

Algorithm 5 p-`dispersion-grasp`(Instance, α_l)

Input: Instance of the problem, α_l quality threshold for the distance

Output: Y solution set of p centers

```
1: $d$, $S$, $p \leftarrow$ parameters from the Instance.
2: $N \leftarrow |S|$
3: Find $(x_i, x_j)$ such that $d_{x_i,x_j} = \max\{d_{x_i,x_j} : 1 \leq 1 < j \leq |S|\}$
4: $Y \leftarrow \{x_i, x_j\}$
5: while $|Y| < p$ do
6:     Define $\phi(x_i) = \min\{d_{x_i,x_j} : x_j \in Y\}$ for $x_i \in \{1, \dots, N\} \setminus Y$
7:     $\phi_{\max} \leftarrow \max\{\phi(x_i) : x_i \in \{1, \dots, N\} \setminus Y\}$
8:     $\phi_{\min} \leftarrow \min\{\phi(x_i) : x_i \in \{1, \dots, N\} \setminus Y\}$
9:     $RCL \leftarrow \{x_i \in \{1, \dots, N\} \setminus Y : \phi(x_i) \geq \phi_{\max} - \alpha_l(\phi_{\max} - \phi_{\min})\}$
10:    Select random $x_i$ from RCL
11:    $Y \leftarrow Y \cup \{x_i\}$
12: end while
13: return $Y$
```

On the other hand, for the allocation phase, instead of choosing the nearest assignment for the BUs in the priority queue, we define a greedy function $\phi(j)$: $\min(d_{i,j}, i \in Y)$ that minimizes the distance to a BU j from all available centers in Y, defining $\phi_{\max}$ as the furthest center to j and $\phi_{\min}$ as the nearest center to j. In this case, due to using precomputed data structures and sorting the centers, $\phi_{\max}$ is the last element in the dictionary values for the best assignments of a BU j, and $\phi_{\min}$ is the first element in such values, as they are sorted already. The RCL is constructed as RCL $\leftarrow \{j : \phi(j) \leq \phi_{\min} - \alpha_a(\phi_{\max} - \phi_{\min})\}$. This is detailed in Algorithm 6.

Algorithm 6 `opp-cost-queue-grasp`(Instance, Y, α_a)

Input: Instance of the problem, Y decision variable, α_a: quality threshold for the allocation RCL

Output: X decision matrix of allocations

```
1: Distance matrix $d$, Decision Vector $Y$, $B$ set of BUs, $S$ set of BUs, $p$ centers to
   be used , $\beta$ risk threshold $\leftarrow$ Instance parameters
2: $pq$, $best\_assignments \leftarrow$ compute-structures$(d, Y, B, S, p)$
3: $assigned\_clients \leftarrow \emptyset$
4: $full\_centers \leftarrow \emptyset$
5: while $pq$ is not empty and $|assigned\_clients| < B$ do
6:     $j \leftarrow$ dequeue $pq$
7:     if $j \notin assigned\_clients$ then
8:         $\phi_{\max} \leftarrow$ last element from $best\_assignments[j]$
9:         $\phi_{\min} \leftarrow$ first element from $best\_assignments[j]$
10:        RCL$\leftarrow \{j : \phi(j) \leq \phi_{min} - \alpha_a(\phi_{max} - \phi_{min})\}$
11:        Select random $i$ from RCL as long as it is not in $full\_centers$
12:        $X_{i,j} \leftarrow 1$
13:        Add client to $assigned\_clients$
14:        Update $values\_m$, $risk\_v$ with $i, j$
15:        if center $i$ has reached lower bounds in activity measures or in the risk
           value then
16:            Add $i$ to $full\_centers$
17:            break
18:        end if
19:    end if
20: end while
21: Assign all clients $\notin assigned\_clients$ to the centers with the largest capacity
    available in the risk threshold $\beta$.
22: return $X$
```

4.3 Local Search Heuristics

Local search heuristics operate on the principle of iteratively exploring the neighborhood of a current solution in the search space to find a new solution that is better according to a given objective function (Taillard, 2023). *The neighborhood* of a solution typically consists of all solutions that have a small, predefined change in their structure from the current solution. The steps taken to move from one solution to another are defined by what is called a *move operator*.

They explore the local space of a given initial solution trying to either improve the objective function value or repair its feasibility of the solution, we two strategies were implemented and tested: (i) A Best Found (BF) strategy that moved to the best improving neighbor and (ii) a First Found (FF) strategy that moved to the first improving neighbor. In our testings, shown in Table 5.4, the FF yield better results, so this is the strategy chosen for the final GRASP implementation.

In this work, some infeasible initial solutions can be a result of the constructive heuristic, because of this, we have developed a Local Search procedure that can handle infeasibility by first repairing the solutions, once solutions are feasible, it moves on improving the solution quality. This procedure is explained below.

4.3.1 Repairing Infeasibilities

As stated previously, the constructive heuristic proposed in this thesis may provide solutions that are not feasible. A solution may be infeasible because of two reasons: a center exceeds the maximum number of allocated BUs, because of the activities' measures surpassing their maximum target, or the risk threshold is exceeded; or the BUs assigned were not enough to complete the minimum required in the activities' measures. To avoid this, the local search applies one of the two following basic moves: **Add**(i) which will add BUs to a center i, that is violating the lower bounds

of the constraints until those bounds are met, while ensuring that the moved BUs do not render their previous centers infeasible, and **Remove**(i) which will remove BUs from a center i that exceeds the upper bounds of the constraints, ensuring that whichever center those BUs end up being reallocated to does not become infeasible as well.

By directing the local search with "AddTo" or "RemoveFrom" sets for centers, we achieve faster computations, as it is a straight-forward to determine to which set does a center is member of, as shown in Algorithm 7. The moves are applied until all centers are feasible, transforming the solution to a feasible one, expressed in Algorithm 8.

Algorithm 7 `compute-add-remove`(Instance, X, Y)

```
Input: Instance of the problem, $X$ decision matrix, $Y$ decision vector
Output: $AddTo$ set of centers to add BUs, $RemoveFrom$ set of centers to remove BUs
1: for $i \in Y$ do
2:     if Any value from $X, Y_i$ is greater than the upper bounds from the constraints then
3:         $RemoveFrom \leftarrow i$
4:     else if Any value from $X, Y_i$ is less than the lower bounds from the constraints then
5:         $AddTo \leftarrow i$
6:     end if
7: end for
8: return $AddTo, RemoveFrom$
```

Algorithm 8 `repair-solution`(Instance, X, Y)

```
Input: Instance of the problem, $X$ decision matrix, $Y$ decision vector
Output: $X'$ decision matrix
 1: $AddTo, RemoveFrom \leftarrow$ compute-add-remove(Instance, $X$, $Y$)
 2: $X'$ copies $X$
 3: for $\tilde{i} \in AddTo$ do
 4:     while Any value from $X, Y_{\tilde{i}}$ is less than the lower bounds from the constraints do
 5:         Allocate the nearest $j$ BU to $\tilde{i}$, removing it from its previous center $i*$
 6:         $X'_{i*,j} = 0$
 7:         $X'_{\tilde{i},j} = 1$
 8:         Update the constraints' states for $\tilde{i}$ and $i*$
 9:     end while
10: end for
11: for $\tilde{i} \in RemoveFrom$ do
12:     while Any value from $X, Y_{\tilde{i}}$ is greater than the upper bounds from the constraints do
13:         Move the furthest $j$ BU from $\tilde{i}$, allocating it to its nearest center $i*$
14:         $X'_{\tilde{i},j} = 0$
15:         $X'_{i*,j} = 1$
16:         Update the constraints' states for $\tilde{i}$ and $i*$
17:     end while
18: end for
19: return $X'$
```

4.3.2 Improving objective function value

Once the solution is feasible, the local search can now proceed to improve the objective function value. For this, the following moves are defined:

- **Allocate Other**(i, j): Reassign a BU j from center i to another center $\tilde{i}$, where $\tilde{i} \neq i$, $\iff X_{ij} = 1$.

- **Interchange**$(i, j, \tilde{i}, \tilde{j})$: Swap allocations of BUs j and $\tilde{j}$ such that j is allocated to $\tilde{i}$ and $\tilde{j}$ to i, $\iff X_{ij} = 1 \wedge X_{\tilde{i}\tilde{j}} = 1$.

- **Deactivate**$(i, \tilde{i})$: Close center i and open an unused center $\tilde{i}$. Assign BUs to $\tilde{i}$ until the lower bounds of the balancing constraints are met, and reallocate the remaining BUs from i to the best-fit centers, $\iff Y_i = 1 \wedge Y_{\tilde{i}} = 0$.

The Deactivate move reallocates the remaining BUs using the same dictionary with the best assignments to each BU from the allocation phase of the constructive heuristic, as it already has the centers sorted by their distance to the BUs.

4.3.3 Combining Multiple Neighborhoods

As we have designed multiple neighborhoods for the local search phase, it is natural to combine these multiple moves in a cyclical procedure, exploring the local space of a move until it reaches the local optimum value and then it moves on to the next move, until no move improves the solution. This helps us escape local optima and provides a more comprehensive exploration of the solution space. The main steps of the algorithm are are:

1. Start with an Initial Solution: Begin with an initial solution and a set of predefined neighborhood structures. In this case, the initial solution is the solution provided by the constructive heuristic and the set of neighborhoods is the collection of the 3 moves defined previously.

2. Sequential Neighborhood Search: Explore these neighborhoods in a systematic way. For each neighborhood, perform a local search to find a local optimum. In this case, we performed the **Allocate to Other, Interchange** and **Deactivate** moves sequentially in that order, applying the moves iteratively to the result of the

previous move until all three neighborhoods reached their local optimum, doing so cyclically.

3. Neighborhood Change: If an improvement is found in any neighborhood, the process is restarted from the first neighborhood with the improved solution. If no improvement is found in any of the neighborhoods, the process stops, and the best-found solution is returned.

In Algorithm 9, lines 1 to 3 repair the solution using the previously defined algorithms. The cyclical strategy is applied as a while loop from lines 6 to 29, applying the moves in the sequential order show, until no move improves the incumbent solution. A solution is expressed as (X, Y) where X is the decision variable matrix of allocations and Y the decision variable vector of locations.

Algorithm 9 `local-search`(Instance, X, Y)

Input: Instance of the problem, X decision matrix, Y decision vector

Output: X decision matrix, Y decision vector

```
1:  if $(X, Y)$ is not feasible then
2:      $X \leftarrow$ repair-solution($(X, Y)$)
3:  end if
4:  improvement $\leftarrow$ True
5:  $z \leftarrow f(X, Y)$
6:  while improvement do
7:      improvement = False
8:      $(X', Y') \leftarrow$ Apply Allocate Other to $(X, Y)$ until local optimum is met
9:      $z' \leftarrow f(X', Y')$
10:     if $z' < z$ then
11:         $(X, Y) \leftarrow (X', Y')$
12:         $z = z'$
13:         improvement = True
14:     end if
15:     $(X', Y') \leftarrow$ Apply Interchange to $(X, Y)$ until local optimum is met
16:     $z' \leftarrow f(X', Y')$
17:     if $z' < z$ then
18:         $(X, Y) \leftarrow (X', Y')$
19:         $z = z'$
20:         improvement = True
21:     end if
22:     $(X', Y') \leftarrow$ Apply Deactivate to $(X, Y)$ until local optimum is met
23:     $z' \leftarrow f(X', Y')$
24:     if $z' < z$ then
25:         $(X, Y) \leftarrow (X', Y')$
26:         $z = z'$
27:         improvement = True
28:     end if
29: end while
30: return $(X, Y)$
```

4.4 Parallel computing

As the individual components for the heuristics were chosen due to their performance detailed in Chapter 5, the GRASP metaheuristic was built from these individual algorithms. An advantage of the GRASP metaheuristic is that each individual iteration can be parallelized, as each iteration is unique and independent from the next or previous one. By assigning an iteration to a computer core thread, we can speed up the computation by introducing parallelization with the use of such computer threads, synchronizing the threads only when checking for the best solution found across the multi-threaded computation. This parallelization mechanism was successfully implemented for the GRASP metaheuristic, providing faster results compared to a single-threaded GRASP version, as shown in section 5.5.1.1.

In order to properly implement a multi-threaded version of the GRASP metaheuristic we had to use the computer science concept of locks. A lock, also known as a mutex (derived from mutual exclusion) is a construct that allows us to safely manage different state variables, by not allowing them to be accessed from multiple threads at the same time, preventing race conditions, which occur when two different threads try to modify the same variable at the same time, entering into a “race” of accessing and modifying the variable, ensuring a safe and correct implementation of the algorithm.

Chapter 5

Computational Results

In this chapter we present the results of the computational experience of the proposed algorithm. We highlight the performance advantages of using data structures and an intelligent design of algorithms versus a more naive version of the algorithm. Additionally, we compare the obtained solutions with those provided by Gurobi version 11.0.3, a state-of-the-art solver. We also show the effectiveness of implementing a parallel version of GRASP, reducing running times significantly. Finally, we provide the best parameters to be used in the GRASP metaheuristics as result of a fine-tuning phase.

5.1 Instance generation

To assess our procedure, some problem instances were randomly generated. Throughout the thesis, the Julia programming language is used to carry out all the computations related to the problem. The Julia language was designed to be used in scientific computing contexts, with a simple syntax reminiscent of Python and FORTRAN while providing the performance of C/C++, showing that one can have machine performance without sacrificing readability (Bezanson et al., 2014). From the instance generation, to the modeling of the problem as a mathematical program and the GRASP metaheuristic, every stage of computation was written in Julia.

In order to generate problem instances, several routines are detailed and executed in the following order: first, the coordinates in a Cartesian plane of the BUs and the centers are randomly generated from an interval of 5 to 10,000; second, the d distance matrix is calculated from these coordinates using the Euclidean distance metric, rounding the distances to integers. Third, the T_{ik} parameter is generated, assigning a random k type to each center, we define the L_k and U_k bounds proportional to the the sum of all the centers for each type k. Afterwards, for each activity measure m, a random value is assigned to every individual BU j in the vector v_m^j, from a range of 10 to 30. Once all the BUs have an activity measure value for all the activities, the targets of the measures are calculated as the sum of all the measure values divided by $|S|$ and multiplied by an additional τ parameter to introduce some leeway, with a value of 1. The t_m tolerance parameter is defined as 0.4. Finally, the risk value for every BU j is randomly generated in the 40 to 60 range, with the risk threshold for each center i defined as the sum of all the individual values divided by $|S|$ and multiplied by the same τ parameter $((\sum_{j \in B} \frac{r_j}{S})\tau)$.

All of the instances are written using an HDF5 standard compliant file format, in order to allow serialization and data sharing between programming languages and computing platforms, as long as the programming language has a valid HDF5 parser library to read the instances and the solutions (Koranne, 2011).

Three sizes of instances were generated, the BUs and centers' coordinates were generated from a range of [5, 10,000] in the Cartesian plane, with the following configuration:

Table 5.1: Sizes of the generated instances

Size	$\|B\|$	$\|S\|$	p	Number of instances
1	625	62	32	20
2	1250	155	62	20
3	2500	325	125	10

5.2 Exact solutions

Algebraic modeling languages (AMLs) are domain specific computer languages that are used to describe and solve mathematical optimization models with a similar syntax to the mathematical notation of optimization problems. This makes it possible to define optimization problems in a very clear and understandable manner. This is facilitated by specific language components such as sets, indices, algebraic expressions, sparse indices, variables for handling data, and constraints with arbitrary names (Kallrath, 2004).

Instead of directly solving those problems, an AML calls relevant external algorithms to optimize the problem. These programs, known as solvers, use exact algorithms designed to handle a wide range of optimization problems, such as linear programs, integer programs, and quadratic programs, for example. The algorithms use mainly branch and bound, and algorithmic strategies depending on the structure of the structure of the problem and the solver itself. These algorithms can also be used to implement more advanced methods such as branch and cut, branch and price, or cutting planes.

By using an AML to represent the mathematical model described in Chapter 3, we can feed the resulting abstraction of the model to an optimization software package.

In this specific case, the algebraic modeling language used is the JuMP.jl package from the Julia language. JuMP enables developers to use a very concise syntax that enables a near one-to-one mapping of the mathematical notation to the language features (Dunning et al., 2017), and the mathematical solver used is the Gurobi Optimizer, a commercial solver that uses state-of-the-art optimization algorithms and techniques to solve problems, providing better results than open source solvers can (Gurobi Optimization, LLC, 2023). We briefly tested the CPLEX and HiGHS optimizers but decided to use Gurobi based on the preliminary results when comparing

to the mentioned solvers.

Using the Julia language version 1.10.4 and Gurobi version 11.0.3, we modeled the problem using JuMP version 1.22.2 and introduced the corresponding model and instance to the solver, setting two time limit for the optimization process of 1800 seconds and 7200 seconds, varying on the size of the instance. The results reported are the total optimization time of the algorithm, the best integer solution found, the best lower bound, as well as the relative optimality gap or MIP (Mixed Integer Programming) gap. According to Gurobi Optimization, LLC (2023), "the MIP gap is defined as $|z_P - z_D|/|z_P$, where z_P is the primal objective bound (i.e., the incumbent objective value, which is the upper bound for minimization problems), and z_D is the dual objective bound (the lower bound for minimization problems)". The MIP gap for the termination of Gurobi's algorithm is left as the default value of 0.01%, a hundredth part of 1%.

For Size 1, the results are shown in Table 5.2. As appreciated, out of the 20 instances, no solutions were proven to be optimal within the time limit of 1800 seconds (half an hour). The average relative optimality gap (MIP gap) percentage of the solutions is 0.28%, proving that even for the smallest instances, the $\mathcal{NP}$ nature of the problem is impacting the performance of exact solvers, although the gap to optimality is minimal. This is useful when comparing to the results of the heuristics, as we can compare to near optimal results provided by the best bound.

As shown in the Appendix in tables A.1 and A.3, for Sizes 2 and 3, no optimal solution was found within half and hour and 2 hours, the respective time limits for the instances. The average MIP gap percentage of Size 2 is 0.45% and for Size 3 is 0.18%. These values represent almost optimality, in most cases, the optimal solution was reached and the algorithm spent the rest of the time proving optimality. Since we are dealing with an $\mathcal{NP}$-hard problem, Gurobi struggles to prove that the best solution is the optimal one, spending computational time, showcasing the usefulness of the heuristics.

Table 5.2: Exact solver solutions for Size 1

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	1800	464274	462154.44	0.46
2	1800	475106	474267.15	0.18
3	1800	451907	451740.15	0.04
4	1800	460877	460520.42	0.08
5	1800	469190	468298.94	0.19
6	1800	491611	490305.18	0.27
7	1800	454550	453366.41	0.26
8	1800	462113	460970.18	0.25
9	1800	488971	487836.91	0.23
10	1800	471810	467610.27	0.89
11	1800	468195	466413.29	0.38
12	1800	461823	461040.14	0.17
13	1800	468480	468091.44	0.08
14	1800	459301	459097.84	0.04
15	1800	483950	479672.39	0.88
16	1800	462365	461967.27	0.09
17	1800	462745	462126.53	0.13
18	1800	455692	454556.37	0.25
19	1800	461881	458774.94	0.67
20	1800	465070	464547.88	0.11

5.3 Constructive heuristics

In this section the results of the different strategies that compose the constructive heuristic are presented. As a reminder, the heuristic has two main phases: location and allocation. For location, we defined that we can use either a p-dispersion based heuristic or the Partial Linear Programming (PLP henceforth) relaxation approach. For allocation, we can use the opportunity cost algorithm that calculates the whole opportunity cost matrix or the algorithm that incorporates data structures such as the opportunity cost priority queue, referring to each as "matrix" or "queue" algorithms, respectively. All instances of Sizes 1 and 2 were solved with all the possible combinations of location and allocation strategies. In Table 5.3 we can see the averages of the run-time of the algorithms and the number of constraints violated by the resulting solutions, which gives us an insight on how far away are the solutions from feasibility. ε denotes a time equal or shorter than 0.01 seconds. The Total Time column is the sum of the location and allocation phases.

Table 5.3: Comparison among the Constructive Heuristic strategies (averages)

Inst. Size	Location Strat.	Location Time (s)	Allocation Time (s)		Total Time (s)		# of violated constraints before repair	
			Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.
1	P-disp	0.02	0.22	ε	0.24	0.04	43.75	5.4
	Relaxation	1.00	0.31	ε	1.31	1.01	49.75	4.6
2	P-disp	0.03	1.60	0.02	1.63	0.05	81.55	0.15
	Relaxation	11.61	1.63	0.02	13.23	11.79	92.01	0.30

The p-dispersion location strategy outperforms the partial linear programming relaxation strategy in execution time, therefore we will use it as the building block for our GRASP metaheuristic. Meanwhile, the queue allocation strategy consistently outperforms the matrix allocation strategy in both run time and feasibility of the created solution. As seen in the table, as the instances get larger, the queue allocation algorithm almost always provides feasible solutions, reducing the need of using the

repair algorithm in the local search procedure.

5.4 Local search heuristics

As defined in the previous chapter, there are three moves to perform in the local search procedure: **Allocate to Other, Interchange** and **Deactivate**. These three moves were used together in a cyclic local search, in the same order, applying the moves iteratively to the result of the previous move until all three neighborhoods reached their local optimum. Local search heuristics can guide their search by either moving to the best solution found across the neighborhood or move to the first solution that improves the objective function value, without exploring all the alternatives. We define these two strategies as Best Found (BF) or First Found (FF). Applying the local search with either BF or FF strategy to the solutions produced by the constructive heuristic we present the following results:

Table 5.4: Evaluation of local search strategies (averages)

Inst. Size	Strategy	% Rel. Optimality Gap	Time (s)
1	BF	10.88	0.89
	FF	15.49	0.52
2	BF	10.73	10.98
	FF	10.52	6.37

As appreciated, the Best Found strategy provides slightly better results on average in the optimality gap to the best bound in the Size 1 instances, but requires a longer time. For Size 2, the gaps are comparable, and times are shorter for First Found. Therefore, we chose the First Found strategy to be used in the GRASP search procedure, preferring shorter run-times.

5.5 GRASP

With the p-dispersion and opportunity cost queue enhanced strategies chosen for our constructive heuristic and the First Found strategy along with the cyclical mechanism for the search heuristic, we can now integrate and adapt these building blocks into the Greedy Randomized Adaptive Search Procedure metaheuristic, defining their respective modifications in Chapter 4.

GRASP requires the α_l and α_a quality threshold parameters for the RCL as well as the number of iterations that it will run for. Moreover, as our GRASP can be parallelized, the number of computer core threads used to parallelize the computation needs to be calibrated as well for maximum performance.

5.5.1 Fine-tuning of GRASP parameters

5.5.1.1 Number of threads

The first parameter that needs to be calibrated is the number of threads that the GRASP metaheuristic will run on. We defined a basic configuration with $\alpha_l = 0.1, \alpha_a = 0.1$ and 50 iterations. As the number of threads depends on the number of cores of each computer's CPU, we decided to run from 1 to 16 threads, as that is the maximum number of threads in the computing platform in which we ran the experiments. We obtained the results shown in Figures 5.1 and 5.2.

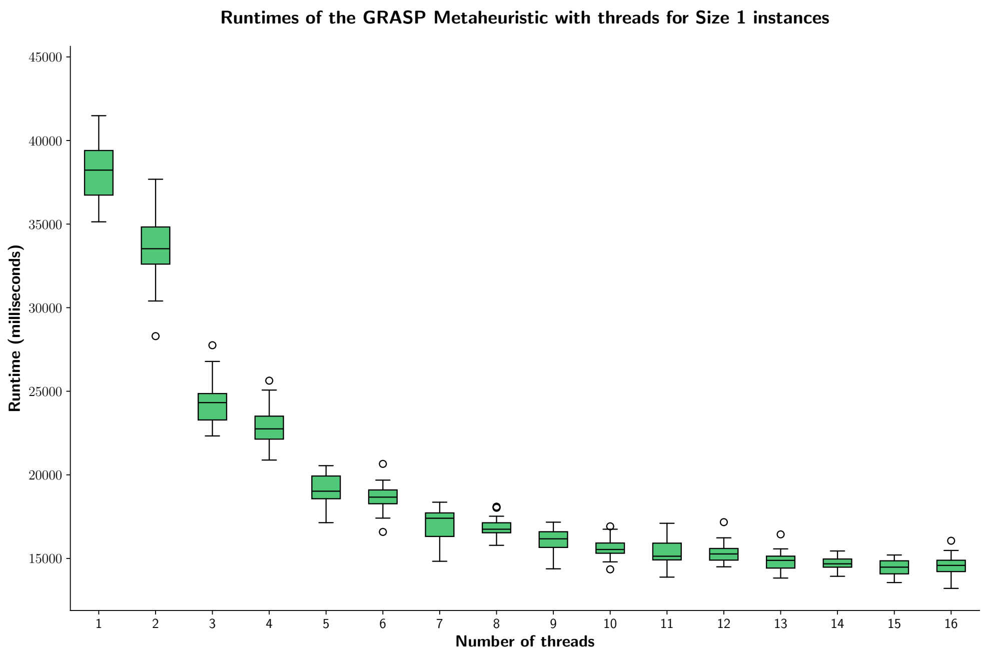


Figure 5.1: Boxplot of the GRASP's run-time for Size 1 instances

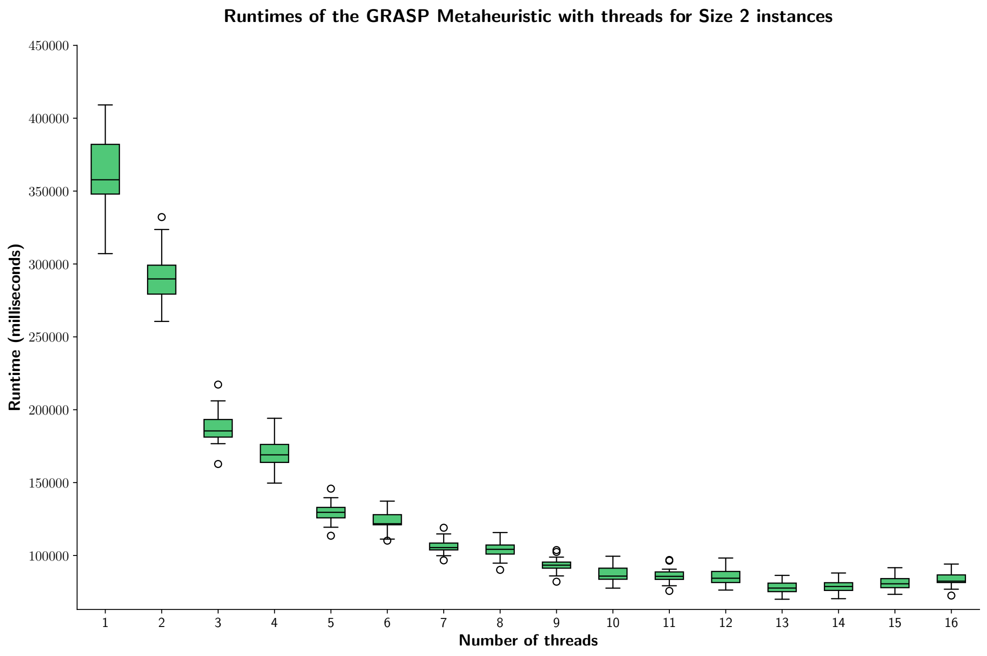


Figure 5.2: Boxplot of the GRASP's run-time for Size 2 instances

Statistical hypothesis testing was performed on the results to determine the optimal amount of threads to use as well as to check if that number of threads had any impact on the objective function value, which is something undesired.

The full extent of the statistical tests is present in the Appendix, more specifically section A.5, the main conclusions extracted were that: increasing the number of threads generally improves performance, with the most substantial gains observed when moving from 1 to 14 threads; the number of threads for minimizing run-time appears to be around 14 or 15; performance plateaus or shows minor reductions after 14 threads, increasing more the number of threads does not have a significant impact on the objective value, as evidenced by the ANOVA (Analysis of Variance) on objective values. After 14 threads, the overhead of communication between the computing processes, the scheduler of the operating system and the memory requirements negate any further improvement in the computation time of the heuristic.

A very interesting thing to note as well is that the size of the problem impacts the performance per thread. For Size 1, we minimize run-times using 16 threads, but due to a heavier computing load for Size 2, we find a better run-time with 14 rather than 16 threads. This can be explained using Amdahl's law (Amdahl, 1967), a computer architecture law that is used in parallel computing to explain and predict the speedup of a process if ran in multiple processors or cores. A heavier load may also impact the task scheduler of the host operating system, larger instances may also incur more memory costs, having a larger overhead, and in this case, making the increasing thread count a penalty from 15 threads onward.

When testing for Size 3 instances, a high variability when running different numbers of threads was found. In some runs, 8 threads provided the best times, whereas for the same instance, if ran with 12 threads at a later point in time, it ran faster than 8 threads, but when trying to replicate even later, the 12 threads were worse than the 8 threads. This was a hindrance on testing and calibrating the number of threads for larger instances, but was an interesting finding, as it confirms

that the size of the instance does impact the best number of threads, pointing to the computing overhead of more complex instances. The reasons of this inconsistent behavior were not made clear, so future work could tackle this issue.

It is important to note that Gurobi will use all available CPU cores by default, making it parallel as well.

5.5.1.2 Calibration of α parameters

Next we have to calibrate the α parameters as well as the number of iterations. We tested the following combinations for all instances of Sizes 1 and 2: $\alpha_l \in \{0.1, 0.3, 0.5, 0.7\}$, $\alpha_a \in \{0.1, 0.3, 0.5, 0.7\}$ and $iters \in \{25, 50, 75, 100\}$, giving us a total of 64 combinations to run. In Tables A.28 and A.29 we present the statistical description of running the GRASP for all the combinations, solving 10 instances of Sizes 1 and 2 each.

We carried out one-way Analysis of Variance (ANOVA) on the three parameters and whether they have an impact or not in the run-time of the metaheuristic as well as the objective function value. For the iterations parameter, the ANOVA test on both results revealed statistically significant differences across the different levels of iterations, meanwhile both α did show statistically significant differences across their different levels for instances of Size 1, but the same behavior was not seen in instances of Size 2. We also carried out a Factorial ANOVA to statistically control for variables and the interactions they may have between each other, obtaining varied results for both instance sizes. As the number of iterations increases, so does the run-time and the objective function improves slightly, but the results for α are inconclusive, as different results are observed across the different sizes, they may also be due to the factorial nature of this test. Finally, we performed Tukey's Honest Significant Differences testing, showing that α_l and α_a have impact on the objective function value for instances of Size 1, but not for instances of Size 2. Therefore, we decided

to settle for $\alpha_l = 0.1, \alpha_a = 0.3$ and $iters = 100$. Both α have good enough averages across all of the combinations for the two instance sizes and the number of iterations decreases our relative optimality gap, incurring a time penalty that is acceptable when comparing to the run-time of Gurobi.

5.5.2 GRASP performance

With $\alpha_l = 0.1, \alpha_a = 0.3$, $iters = 100$ and the number of threads to be used set to 14, we can now properly run the GRASP metaheuristic and obtain results for all of the instances. In this final experiment, we also used 10 instances of Size 3 to evaluate both the metaheuristic as well as Gurobi to their limit. Due to thermal throttling and inconsistent core usage, we had to reduce the number of iterations to 25 in the instances of Size 3. In this case, Gurobi was set a time limit of two hours, and the results can be found in the Appendix, on Table A.3

The most important table that can condense the entire work of this thesis is the following:

Table 5.5: Results of the Fine-tuned GRASP with 14 threads

Instance Size	% Optimality Gap			Speedup Factor vs Gurobi		
	Min	Avg	Max	Min	Avg	Max
1	5.22	6.71	8.56	82.68	124.74	144
2	6.33	7.55	9.91	11.05	12.25	13.45
3	7.38	8.34	8.97	3.84	4.26	4.95

The optimality gap to Gurobi's best bound z_D is calculated with the same formula defined by Gurobi Optimization, LLC (2023), which in this case z_P is the value provided by the GRASP metaheuristic.

As the table shows, the GRASP metaheuristic runs several orders of magnitude

faster than Gurobi while providing solutions that are a single digit of relative difference with respect to the solutions found by Gurobi. For all the tables of results, when comparing to Gurobi, the relative difference is how worse are the solutions found by the heuristics.

For Instance Size 1, GRASP runs more than 100 times faster than Gurobi, providing solutions that have a 6.71% of optimality gap.

For Instance Size 2, the speedup is not as dramatic but it is still on average 12 times faster, with a slightly worse average optimality gap of 7.55%.

Finally, for Instance Size 3, the speedups are still there, reduced, but still 4 to 5 times faster than Gurobi, with the optimality gap still a digit away. In this final instance size, Gurobi had several out-of-memory errors when working with the instances, so multiple retries had to be done to solve the problem until it reached the specified timeout.

The full results are present in the following tables:

Table 5.6: GRASP results on Size 1 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	1800	464274	0.46	462154.4	15.32	501167	7.78	117.49
2	1800	475106	0.18	474267.2	13.33	507463	6.54	135.03
3	1483	451907	0.04	451740.2	13.48	491261	8.04	133.53
4	1800	460877	0.08	460520.4	13.63	498939	7.70	132.06
5	1800	469190	0.19	468298.9	14.24	496673	5.71	126.40
6	1800	491611	0.27	490305.2	14.90	522147	6.10	120.81
7	1800	454550	0.26	453366.4	12.50	495818	8.56	144.00
8	1800	462113	0.25	460970.2	12.54	497735	7.39	143.54
9	1800	488971	0.23	487836.9	13.23	517519	5.74	136.05
10	1800	471810	0.89	467610.3	21.77	506917	7.75	82.68
11	1800	468195	0.38	466413.3	14.86	507179	8.04	121.13
12	1800	461823	0.17	461040.1	14.91	489697	5.85	120.72
13	1800	468480	0.08	468091.4	13.93	499595	6.31	129.22
14	976	459301	0.04	459097.8	14.11	499018	8.00	127.57
15	1800	483950	0.88	479672.4	14.35	506120	5.23	125.44
16	1384	462365	0.09	461967.3	13.12	496302	6.92	137.20
17	1800	462745	0.13	462126.5	14.88	492972	6.26	120.97
18	1800	455692	0.25	454556.4	14.11	480927	5.48	127.57
19	1800	461881	0.67	458774.9	14.51	485495	5.50	124.05
20	1800	465070	0.11	464547.9	14.88	490130	5.22	120.97

Table 5.7: GRASP results on Size 2 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	1800	949005	0.44	944849.5	145.24	1022786	7.62	12.39
2	1800	975979	0.92	967012.3	140.51	1043541	7.33	12.81
3	1800	988670	0.67	982048.4	159.46	1082240	9.26	11.29
4	1800	974186	0.29	971396.9	147.49	1049803	7.47	12.20
5	1800	958455	0.05	957950.0	135.87	1039603	7.85	13.25
6	1800	964088	0.22	961976.2	139.55	1036826	7.22	12.90
7	1800	968567	0.56	963149.9	143.17	1039149	7.31	12.57
8	1800	986315	0.32	983207.5	133.83	1049604	6.33	13.45
9	1800	951731	0.45	947419.2	139.09	1025924	7.65	12.94
10	1800	999440	0.84	991050.3	159.94	1100104	9.91	11.25
11	1800	961532	0.70	954783.3	137.66	1022340	6.61	13.08
12	1800	942864	0.20	940956.3	152.63	1008988	6.74	11.79
13	1800	1012710	0.32	1009464.0	162.79	1114006	9.38	11.06
14	1800	987762	0.28	985021.3	153.46	1058794	6.97	11.73
15	1800	961675	0.48	957101.4	148.60	1030436	7.12	12.11
16	1800	990828	0.70	983915.7	160.70	1057756	6.98	11.20
17	1800	992812	0.13	991539.1	145.73	1067347	7.10	12.35
18	1800	980761	0.64	974474.1	141.49	1056380	7.75	12.72
19	1800	973562	0.28	970867.4	136.73	1040261	6.67	13.16
20	1800	972605	0.45	968213.7	154.64	1049519	7.75	11.64

Table 5.8: GRASP results on Size 3 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	7200	883693	0.29	881117.8	1746.70	959509	8.17	4.12
2	7200	893408	0.15	892105.5	1655.75	963215	7.38	4.35
3	7200	882602	0.04	882275.8	1452.81	966988	8.76	4.96
4	7200	913347	0.26	910961.7	1748.65	1000684	8.97	4.12
5	7200	895581	0.21	893657.9	1633.76	975003	8.34	4.41
6	7200	896157	0.18	894518.5	1873.31	966452	7.44	3.84
7	7200	893684	0.13	892487.9	1626.62	979920	8.92	4.43
8	7200	895125	0.28	892662.6	1641.34	974882	8.43	4.39
9	7200	877510	0.12	876485.6	1836.52	957017	8.41	3.92
10	7200	892898	0.14	891652.1	1657.39	975116	8.56	4.34

Chapter 6

Conclusions

6.1 Main findings and conclusions

By using intelligent algorithms as well as efficient data structures, we solved the Territorial Design Problem for microfinancial institutions using a GRASP metaheuristic that proved to be competitive with commercial exact optimization software, being several orders of magnitude faster in the execution time while providing solutions that are just slightly worse than the ones found by Gurobi, which are near-optimal.

We developed a mathematical model based on previous work by López, Ekin, Mediavilla, and Jimenez (2020), generalizing it and making it easier to solve in general, building an instances generator program for this model. We successfully implemented this mathematical model in a modeling language in order to be solved with an optimization solver, in this case Gurobi, which is a commercial software that we could use through the use of an academic license.

Using the p-dispersion heuristic for location and opportunity cost with the usage of efficient data structures for allocation we can offer the best performance in feasibility and speed, as well as using a local search heuristic to repair to feasibility the solutions constructed and improve these solutions. These two heuristics were adapted and integrated within the GRASP using a value-based restricted candidate

list to promote diversification. We also reduced the time complexity of the first allocation heuristic by introducing several data structures: a priority queue, a dictionary, arrays and matrices to remove repeated computations as well as partially evaluating the solution built, instead of computing everything every time.

By also using parallel computing, we reduced the times of the metaheuristic, exploiting its parallelizable nature and using modern computing infrastructure that maximizes the performance of today's processors, although for larger instances we found erratic loads in the CPU and the threads, which could be due to thermal throttling, garbage collection of the programming language or frequent cache misses which led to memory access slowdowns.

For small size instances, the heuristics sometimes struggle to find feasible solutions, for larger size instances, feasibility is achieved for all instances. However, once the heuristics are integrated in the GRASP framework, feasibility is always reached.

The results found during the computational experiments enabled us to take informed decisions while building the metaheuristic framework, allowing us to fine-tune the parameters of the GRASP algorithm to provide results with a good balance of runtime and objective function value. GRASP's solutions closely rival Gurobi's but are significantly faster.

6.2 Future work

Instead of using risk as a given parameter of the problem, we could model it as an uncertainty measure of a probability distribution that describes the probability of clients defaulting on their loans and financial services provided. By doing so, we can introduce a stochastic component to the problem, turning it into a stochastic optimization problem, in which we must generate scenarios and use estimations and samples to solve it using a robust framework, according to Spall (2003).

As stated back in Chapter 3, the problem studied in this thesis does not consider contiguity constraints. By incorporating contiguity constraints we can model the problem as a graph, more so than the current problem, and use graph theory algorithms to ensure connected territories as graphs.

We can also try to profile and run the metaheuristic controlling for certain variables to determine the cause of the extreme spikes and valleys in CPU load and temperature when running the multi-threaded GRASP in instances of Size 3. By profiling the code, its memory allocations and setting specific parameters such as voltage of the CPU and keeping track of CPU load and temperature, we can pinpoint where is the issue, allowing us to have a clear answer and a solution to the inconsistent speeds of the GRASP implementation.

Both completely change the nature of the problem as well as requiring that the solution methods are adapted to account for these changes. Thus, they provide interesting avenues of research and future work.

The local search heuristic was improved by using a cyclical approach, laying the groundwork for using a Variable Neighborhood Search metaheuristic (Hansen et al., 2010), which has been successfully implemented for territorial design problems (Quevedo-Orozco and Ríos-Mercado, 2015).

Finally, we can explore the use of other strategies to solve the problem, such as using different metaheuristics: the mentioned VNS, Tabu Search, or Scatter Search. We could even hybridize the heuristic with mathematical programming, by partially solving the problem with the heuristics and then passing it to the solver as a warm start for its algorithms.

Appendix A

Appendix

A.1 Exact solutions tables of results

For the results of Size 1, the table is found in Chapter 5.

Table A.1: Exact solver solutions for Size 2 (Instances 1-10)

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	1800	949005	944849.51	0.44
2	1800	975979	967012.29	0.92
3	1800	988670	982048.42	0.67
4	1800	974186	971396.86	0.29
5	1800	958455	957950.01	0.05
6	1800	964088	961976.21	0.22
7	1800	968567	963149.90	0.56
8	1800	986315	983207.46	0.32
9	951731	951731	947419.24	0.45
10	1800	999440	991050.29	0.84

Table A.2: Exact solver solutions for Size 2 (Instances 11-20)

Instance	Time (seconds)	z	Best Bound	% MIP Gap
11	1800	961532	954783.34	0.70
12	1800	942864	940956.26	0.20
13	1800	1012710	1009464.27	0.32
14	1800	987762	985021.26	0.28
15	1800	961675	957101.37	0.48
16	1800	990828	983915.69	0.70
17	1800	992812	991539.11	0.13
18	1800	980761	974474.12	0.64
19	1800	973562	970867.41	0.28
20	1800	972605	968213.68	0.45

Table A.3: Exact solver solutions for Size 3

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	7200	883693	881117.84	0.29
2	7200	893408	892105.54	0.15
3	7200	882602	882275.84	0.04
4	7200	913347	910961.69	0.26
5	7200	895581	893657.94	0.21
6	7200	896157	894518.51	0.18
7	7200	893684	892487.89	0.13
8	7200	895125	892662.59	0.28
9	7200	877510	876485.60	0.12
10	7200	892898	891652.12	0.14

A.2 Constructive heuristic tables of results

A very interesting thing to notice is that due to the just-in-time compilation of the Julia language, the first time a function is called, it is also compiled. For all sizes, the 10th instance was the first to be solved, thus all the times for Instance 10 include the compilation time, therefore all times will show up as outliers when compared to the other solutions' times. Still, the compilation time is minimal and has no major impact on results.

The ε parameter denotes a time shorter than 0.1 seconds. The location phase time is reported as 1.0 seconds because that is the minimum time reported by the optimizer in the Partial Linear Programming Relaxation approach.

A.2.1 Instances of Size 1

Table A.4: Results of constructive heuristics under Opp. Queue and PLP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	1	ε	False	4
2	1	ε	False	3
3	1	ε	False	3
4	1	ε	False	3
5	1	ε	False	3
6	1	ε	False	3
7	1	ε	False	3
8	1	ε	False	7
9	1	ε	False	4
10	1	0.27	False	3
11	1	ε	False	7
12	1	ε	False	6
13	1	ε	False	5
14	1	ε	False	6
15	1	ε	False	3
16	1	ε	False	10
17	1	ε	False	3
18	1	ε	False	6
19	1	ε	False	6
20	1	ε	False	4

Table A.5: Results of constructive heuristics under Opp. Queue and p-dP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	ε	False	6
2	ε	ε	False	8
3	ε	ε	False	7
4	ε	ε	False	7
5	ε	ε	False	5
6	ε	ε	False	3
7	ε	ε	False	3
8	ε	ε	False	6
9	ε	ε	False	3
10	0.55	0.27	False	6
11	ε	ε	False	3
12	ε	ε	False	6
13	ε	ε	False	9
14	ε	ε	False	6
15	ε	ε	False	6
16	ε	ε	False	3
17	ε	ε	False	3
18	ε	ε	False	6
19	ε	ε	False	5
20	ε	ε	False	7

Table A.6: Results of constructive heuristics under Opp. Matrix and PLP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	1	0.37	False	45
2	1	0.36	False	51
3	1	0.17	False	46
4	1	0.36	False	49
5	1	0.17	False	50
6	1	0.40	False	49
7	1	0.17	False	46
8	1	0.46	False	49
9	1	0.16	False	43
10	1	0.71	False	51
11	1	0.50	False	53
12	1	0.21	False	55
13	1	0.43	False	50
14	1	0.22	False	54
15	1	0.21	False	48
16	1	0.38	False	53
17	1	0.16	False	54
18	1	0.38	False	53
19	1	0.19	False	49
20	1	0.18	False	47

Table A.7: Results of constructive heuristics under Opp. Matrix and p-dP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	0.21	False	42
2	ε	0.12	False	41
3	ε	0.12	False	38
4	ε	0.12	False	47
5	ε	0.15	False	37
6	ε	0.13	False	49
7	ε	0.15	False	44
8	ε	0.15	False	41
9	ε	0.17	False	43
10	0.55	0.68	False	44
11	ε	0.25	False	40
12	ε	0.20	False	43
13	ε	0.24	False	46
14	ε	0.23	False	49
15	ε	0.24	False	45
16	ε	0.20	False	47
17	ε	0.20	False	44
18	ε	0.17	False	43
19	ε	0.23	False	42
20	ε	0.38	False	50

A.2.2 Instances of Size 2

Table A.8: Results of constructive heuristics under Opp. Queue and PLP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	12	ε	True	0
2	11	ε	True	0
3	15	ε	True	0
4	13	ε	True	0
5	10	ε	False	3
6	10	ε	True	0
7	11	ε	True	0
8	13	ε	True	0
9	11	ε	True	0
10	15	0.29	True	0
11	11	ε	True	0
12	12	ε	True	0
13	12	ε	True	0
14	11	ε	True	0
15	12	ε	True	0
16	11	0.05	False	3
17	11	ε	True	0
18	12	ε	True	0
19	11	ε	True	0
20	11	ε	True	0

Table A.9: Results of constructive heuristics under Opp. Queue and p-dP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	ε	True	0
2	ε	0.01	False	1
3	ε	ε	True	0
4	ε	ε	True	0
5	ε	ε	True	0
6	ε	ε	True	0
7	ε	ε	False	2
8	ε	ε	True	0
9	ε	ε	True	0
10	0.56	0.28	True	0
11	ε	ε	True	0
12	ε	ε	True	0
13	ε	ε	True	0
14	ε	ε	True	0
15	ε	ε	True	0
16	ε	ε	True	0
17	ε	0.01	True	0
18	ε	ε	True	0
19	ε	ε	True	0
20	ε	ε	True	0

Table A.10: Results of constructive heuristics under Opp. Matrix and PLP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	12	1.57	False	88
2	11	1.53	False	103
3	15	1.5	False	98
4	12	1.92	False	93
5	10	1.38	False	81
6	10	2.02	False	94
7	11	1.1	False	96
8	13	1.37	False	91
9	11	1.75	False	99
10	15	1.92	False	98
11	11	1.42	False	92
12	12	2.13	False	85
13	12	1.16	False	101
14	10	1.69	False	87
15	12	1.64	False	82
16	11	2.0	False	101
17	10	1.64	False	83
18	12	1.28	False	85
19	11	2.08	False	89
20	11	1.52	False	94

Table A.11: Results of constructive heuristics under Opp. Matrix and p-dP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	1.5	False	87
2	ε	1.79	False	78
3	ε	1.47	False	80
4	ε	1.77	False	79
5	ε	1.25	False	84
6	ε	1.56	False	80
7	ε	1.87	False	69
8	ε	1.64	False	76
9	ε	1.07	False	80
10	0.54	1.84	False	90
11	ε	1.6	False	73
12	ε	1.6	False	89
13	ε	1.73	False	86
14	ε	1.6	False	73
15	ε	1.6	False	81
16	ε	1.56	False	84
17	ε	1.72	False	90
18	ε	1.36	False	80
19	ε	1.79	False	81
20	ε	1.71	False	91

A.3 Local search heuristic tables of results

The ε parameter denotes a time shorter than 0.1 seconds, LS Cycles is the number of times the Local Search was performed in a cyclic way, first performing a move, moving to another one when the local optimum was reached, while any move presented an improvement over the incumbent best solution. Instances with a relative optimality gap of 100% are instances that could not be repaired to feasibility by the heuristic.

A.3.1 Instances of Size 1

Table A.12: Local Search results with Matrix, PLP and BF Strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.43	0.7	4	83.28	10.54
2	False	1.7	ε	0	0	100
3	True	0.67	0.84	4	132.84	11.75
4	False	1.87	ε	0	0	100
5	True	0.6	0.65	4	63.16	16.73
6	False	2.18	ε	0	0	100
7	False	1.71	ε	0	0	100
8	False	1.81	ε	0	0	100
9	True	0.65	0.72	5	55.01	15.14
10	True	0.63	0.75	5	59.69	7.55
11	False	1.81	ε	0	0	100
12	False	2.18	ε	0	0	100
13	True	0.65	0.56	5	51.65	10.14
14	True	0.67	0.56	4	42.0	15.37
15	True	0.88	0.69	4	90.31	7.66
16	True	0.62	0.67	5	64.56	6.36
17	True	1.02	0.72	5	80.69	12.0
18	True	1.22	1.09	7	151.49	10.09
19	True	1.08	0.72	5	77.63	8.1
20	False	1.46	ε	0	0	100

Table A.13: Local Search results with Queue, PLP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.42	0.94	6	97.14	10.35
2	True	0.14	1.14	6	155.57	9.84
3	True	0.13	1.27	5	137.57	12.14
4	True	0.1	0.86	4	88.66	15.51
5	True	0.05	1.07	4	86.66	11.14
6	True	0.1	0.81	5	79.8	8.76
7	True	0.14	1.49	4	104.18	10.9
8	False	0.37	ε	0	0	100
9	True	0.35	1.09	5	92.59	14.14
10	False	0.5	ε	0	0	100
11	False	0.83	ε	0	0	100
12	False	0.52	ε	0	0	100
13	True	0.36	0.68	5	71.65	9.05
14	True	0.17	0.77	5	53.07	16.4
15	True	0.24	1.1	7	108.33	10.2
16	False	0.79	ε	0	0	100
17	True	0.09	0.94	5	86.54	12.54
18	False	0.8	ε	0	0	100
19	True	0.08	0.74	4	65.49	12.54
20	True	0.15	0.77	4	91.43	12.3

Table A.14: Local Search results with Matrix, p-dP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.75	1.08	4	94.72	11.93
2	False	2.1	ε	0	0	100
3	True	0.32	0.57	3	47.01	11.82
4	True	0.22	0.58	5	28.64	9.93
5	True	2.05	0.6	5	36.76	10.21
6	True	0.4	1.0	5	81.2	6.34
7	True	0.32	0.64	4	34.45	11.25
8	False	1.57	ε	0	0	100
9	True	0.35	0.77	5	35.12	8.09
10	False	2.13	ε	0	0	100
11	False	1.48	ε	0	0	100
12	True	0.46	0.71	5	38.0	11.95
13	True	0.65	1.01	7	56.05	8.61
14	True	0.53	1.12	5	57.47	12.08
15	True	0.48	1.06	7	74.81	7.98
16	True	0.41	0.72	4	58.18	9.78
17	True	0.58	1.0	4	76.76	11.05
18	True	0.45	0.81	4	51.16	9.02
19	True	0.53	0.76	4	57.19	16.88
20	True	2.58	0.77	5	69.58	5.34

Table A.15: Local Search results with Queue, p-dP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	False	0.43	ε	0	0	100
2	False	0.47	ε	0	0	100
3	True	0.22	0.96	4	100.07	10.06
4	True	0.17	0.76	4	65.4	11.95
5	True	0.23	0.99	4	87.99	6.89
6	True	0.14	1.29	5	108.61	7.48
7	True	0.07	0.88	4	75.35	8.7
8	True	0.23	0.8	4	65.71	8.59
9	True	0.13	1.22	4	94.89	8.65
10	False	0.71	ε	0	0	100
11	False	0.45	ε	0	0	100
12	True	0.18	1.02	6	69.52	10.82
13	False	0.54	ε	0	0	100
14	True	0.2	1.01	3	82.39	14.51
15	True	0.24	1.28	5	95.25	11.87
16	False	0.43	ε	0	0	100
17	True	0.12	1.09	5	101.59	13.55
18	True	0.23	1.11	4	83.32	9.87
19	True	0.07	1.1	4	105.36	16.18
20	False	0.5	ε	0	0	100

Table A.16: Local Search results with Matrix, PLP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.47	0.58	6	86.21	9.11
2	False	1.72	ε	0	0	100
3	True	0.65	0.65	6	136.82	10.24
4	False	1.91	ε	0	0	100
5	True	0.56	0.57	5	71.45	12.49
6	False	2.22	ε	0	0	100
7	False	1.68	ε	0	0	100
8	False	1.86	ε	0	0	100
9	True	0.62	0.4	4	51.49	17.07
10	True	0.64	0.53	5	58.32	8.34
11	False	1.82	ε	0	0	100
12	False	2.18	ε	0	0	100
13	True	0.63	0.4	5	50.6	10.76
14	True	0.65	0.39	4	47.14	12.31
15	True	0.88	0.61	6	83.38	11.02
16	True	0.68	0.42	4	64.29	6.52
17	True	0.97	0.4	4	80.89	11.9
18	True	1.28	0.59	5	154.55	8.99
19	True	1.04	0.51	5	72.94	10.53
20	False	1.45	ε	0	0	100

Table A.17: Local Search results with Queue, PLP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.2	0.41	4	69.72	22.82
2	True	0.11	0.28	3	75.38	38.13
3	True	0.09	0.34	3	79.91	33.47
4	True	0.08	0.64	5	35.05	39.52
5	True	0.07	0.34	3	60.43	23.62
6	True	0.08	0.56	3	47.37	25.22
7	True	0.15	0.33	3	73.59	24.25
8	False	0.59	ε	0	0	100
9	True	0.33	0.25	4	53.07	31.76
10	False	0.48	ε	0	0	100
11	False	0.63	ε	0	0	100
12	False	0.51	ε	0	0	100
13	True	0.12	0.51	3	37.38	27.21
14	True	0.17	0.6	7	35.98	25.74
15	True	0.24	0.42	2	27.63	44.99
16	False	0.51	ε	0	0	100
17	True	0.11	0.34	3	50.89	29.25
18	False	0.59	ε	0	0	100
19	True	0.06	0.39	4	48.21	21.67
20	True	0.14	0.4	2	21.86	44.17

Table A.18: Local Search results with Matrix, p-dP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.75	0.59	4	96.41	11.16
2	False	2.11	ε	0	0	100
3	True	0.32	0.33	4	41.63	15.05
4	True	0.23	0.43	5	27.22	10.92
5	True	2.05	0.5	4	40.1	8.02
6	True	0.4	0.69	5	80.41	6.75
7	True	0.32	0.39	4	36.99	9.57
8	False	1.58	ε	0	0	100
9	True	0.35	0.7	6	38.8	5.58
10	False	2.13	ε	0	0	100
11	False	1.47	ε	0	0	100
12	True	0.46	0.59	6	36.82	12.7
13	True	0.66	0.52	4	45.92	14.54
14	True	0.53	0.67	5	58.28	11.63
15	True	0.47	0.73	6	76.06	7.33
16	True	0.39	0.5	4	62.88	7.1
17	True	0.58	0.64	6	81.71	8.56
18	True	0.45	0.62	5	53.05	7.88
19	True	0.52	0.49	4	58.16	16.37
20	True	2.58	0.56	5	68.62	5.88

Table A.19: Local Search results with Queue, p-dP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	False	0.43	ε	0	0	100
2	False	0.48	ε	0	0	100
3	True	0.22	0.55	4	96.7	11.58
4	True	0.18	0.52	4	68.19	10.47
5	True	0.23	0.56	5	84.28	8.73
6	True	0.14	0.66	4	113.09	5.49
7	True	0.08	0.49	3	75.21	8.77
8	True	0.22	0.53	5	64.51	9.25
9	True	0.13	0.54	4	96.25	8.02
10	False	0.72	ε	0	0	100
11	False	0.43	ε	0	0	100
12	True	0.18	0.64	5	74.78	8.05
13	False	0.54	ε	0	0	100
14	True	0.2	0.56	4	80.09	15.59
15	True	0.24	0.78	6	95.25	11.87
16	False	0.43	ε	0	0	100
17	True	0.1	0.88	6	109.43	10.19
18	True	0.21	0.57	5	89.44	6.86
19	True	0.09	0.74	5	103.97	16.75
20	False	0.51	ε	0	0	100

A.3.2 Instances of Size 2

Table A.20: Local Search results with Matrix, PLP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	5.73	7.92	5	77.64	13.51
2	True	10.97	10.47	5	114.95	11.36
3	True	8.86	8.09	5	71.05	14.36
4	True	5.35	8.73	5	65.89	13.11
5	True	9.84	7.51	4	72.25	9.8
6	True	7.57	9.99	5	88.57	10.78
7	True	4.74	6.67	5	53.97	12.44
8	True	4.99	6.72	6	42.42	10.86
9	True	9.38	8.44	4	91.65	11.02
10	True	10.77	9.5	5	81.07	11.99
11	True	8.79	8.58	6	98.37	8.43
12	True	8.12	6.65	4	69.29	14.16
13	True	7.5	13.07	5	116.47	13.08
14	True	4.46	7.2	5	65.05	10.73
15	True	3.03	5.16	4	18.67	9.75
16	True	6.45	7.25	4	67.63	10.06
17	True	4.27	6.94	5	38.03	8.25
18	True	5.2	7.51	5	57.29	11.23
19	True	5.19	8.57	5	64.51	10.23
20	True	5.7	8.14	4	79.64	8.6

Table A.21: Local Search results with Matrix, p-dP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	2.13	6.26	4	27.3	8.56
2	True	4.35	10.01	5	58.88	8.02
3	True	8.15	13.28	5	68.24	11.25
4	True	3.47	9.37	5	46.52	9.94
5	True	2.82	7.2	4	31.95	10.7
6	True	3.29	7.87	5	40.04	8.44
7	True	3.72	9.33	4	46.89	9.15
8	True	3.35	8.64	5	39.06	13.36
9	True	5.0	10.57	4	56.92	12.4
10	True	4.88	9.28	6	36.41	14.37
11	True	4.09	9.48	5	51.21	8.88
12	True	6.1	9.45	4	59.81	11.16
13	True	3.83	12.53	5	49.78	13.03
14	True	3.07	9.52	5	54.22	7.8
15	True	2.63	9.01	6	52.13	8.72
16	True	5.99	11.46	6	60.72	9.96
17	True	2.69	10.07	5	46.67	9.87
18	True	3.14	8.76	5	45.84	10.74
19	True	3.08	6.81	4	37.12	10.69
20	True	7.81	11.72	4	73.31	10.68

Table A.22: Local Search results with Queue, PLP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	10.3	5	140.11	9.79
2	True	ε	13.14	5	151.9	12.47
3	True	ε	9.91	4	130.8	8.99
4	True	ε	10.47	5	110.77	15.82
5	True	0.03	9.2	4	107.28	8.89
6	True	ε	10.75	6	134.83	11.82
7	True	ε	8.59	4	99.17	10.91
8	True	ε	8.64	6	78.09	10.7
9	True	ε	12.81	5	148.45	12.73
10	True	ε	12.03	5	118.41	13.04
11	True	ε	11.94	5	153.94	6.2
12	True	ε	8.19	4	92.86	14.6
13	True	ε	14.93	5	195.0	8.39
14	True	ε	10.67	5	105.73	10.92
15	True	ε	9.55	5	68.18	9.93
16	True	0.15	10.81	5	114.79	7.97
17	True	ε	11.34	5	95.16	8.38
18	True	ε	8.57	5	97.3	11.51
19	True	ε	13.04	5	108.97	12.26
20	True	ε	10.13	5	116.31	9.08

Table A.23: Local Search results with Queue, p-dP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	12.59	4	104.29	8.87
2	True	0.1	15.86	5	119.76	10.18
3	True	ε	17.05	5	110.68	10.68
4	True	ε	17.2	6	122.46	8.55
5	True	ε	12.68	4	93.14	13.91
6	True	ε	14.55	5	116.24	9.14
7	True	0.02	16.05	4	115.43	9.6
8	True	ε	14.09	4	108.17	9.81
9	True	ε	16.26	5	119.09	12.1
10	True	ε	16.22	5	128.94	11.44
11	True	ε	14.26	4	123.93	8.47
12	True	ε	14.41	5	136.5	11.63
13	True	ε	19.25	5	137.3	12.16
14	True	ε	16.48	5	123.47	11.84
15	True	ε	15.17	5	147.18	9.98
16	True	ε	16.81	5	133.85	11.07
17	True	ε	15.81	5	122.46	8.68
18	True	ε	13.5	4	114.57	11.35
19	True	ε	15.51	5	124.4	11.24
20	True	ε	15.79	5	121.35	11.86

Table A.24: Local Search results with Matrix, PLP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	5.73	4.45	4	75.6	14.5
2	True	11.57	5.72	5	114.32	11.62
3	True	8.93	5.62	4	70.51	14.63
4	True	5.64	5.78	4	66.91	12.58
5	True	10.04	5.14	5	66.59	12.77
6	True	7.54	5.97	5	84.18	12.86
7	True	4.65	4.96	5	55.78	11.41
8	True	5.01	4.77	5	43.86	9.96
9	True	8.81	5.17	5	89.65	11.95
10	True	10.63	6.05	5	76.24	14.33
11	True	8.01	5.29	5	98.34	8.44
12	True	8.9	4.57	5	68.2	14.71
13	True	7.04	7.71	5	126.24	9.15
14	True	4.82	4.6	4	61.63	12.58
15	True	3.07	3.75	4	20.84	8.1
16	True	6.83	6.43	6	67.83	9.95
17	True	4.28	4.86	5	40.83	6.38
18	True	5.12	4.74	4	54.98	12.53
19	True	5.53	5.82	6	61.24	12.01
20	True	5.43	5.84	5	78.65	9.11

Table A.25: Local Search results with Matrix, p-dP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	2.15	4.3	5	26.75	8.95
2	True	4.35	6.15	5	53.53	11.11
3	True	8.03	7.6	5	63.85	13.57
4	True	3.48	5.59	6	45.72	10.43
5	True	2.81	4.28	5	28.34	13.14
6	True	3.27	4.2	4	36.84	10.53
7	True	3.77	6.29	6	48.22	8.33
8	True	3.37	4.68	5	40.54	12.44
9	True	5.02	6.49	6	65.82	7.43
10	True	4.83	5.25	5	37.42	13.73
11	True	4.08	6.15	6	55.38	6.38
12	True	6.07	5.0	4	57.21	12.61
13	True	3.85	6.77	5	50.4	12.67
14	True	3.06	5.27	5	53.78	8.07
15	True	2.62	5.96	7	53.34	8.0
16	True	6.03	7.67	6	60.25	10.22
17	True	2.7	5.57	5	46.39	10.04
18	True	3.15	5.44	5	46.36	10.42
19	True	3.04	4.33	4	35.84	11.52
20	True	7.82	7.59	6	72.14	11.28

Table A.26: Local Search results with Queue, PLP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	5.92	5	141.92	9.11
2	True	ε	6.78	7	152.93	12.12
3	True	ε	7.2	6	118.29	13.92
4	True	ε	7.4	5	121.3	11.61
5	True	0.03	5.32	4	109.84	7.77
6	True	ε	7.49	5	133.86	12.18
7	True	ε	5.34	6	100.44	10.34
8	True	ε	4.71	5	79.94	9.78
9	True	ε	6.61	6	160.4	8.53
10	True	ε	6.41	5	118.95	12.82
11	True	ε	7.21	5	150.7	7.4
12	True	ε	4.95	4	101.45	10.8
13	True	ε	8.48	4	188.72	10.33
14	True	ε	5.56	4	101.75	12.64
15	True	ε	5.93	5	70.52	8.68
16	True	0.3	6.8	5	110.28	9.91
17	True	ε	5.89	5	91.29	10.2
18	True	ε	6.19	6	105.77	7.71
19	True	ε	7.5	5	117.24	8.79
20	True	ε	6.75	6	114.88	9.68

Table A.27: Local Search results with Queue, p-dP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	6.89	4	104.38	8.83
2	True	0.1	7.8	4	122.31	9.14
3	True	ε	9.05	4	104.6	13.26
4	True	ε	8.96	6	120.51	9.35
5	True	ε	6.59	5	96.45	12.44
6	True	ε	7.66	5	118.16	8.33
7	True	0.02	7.9	5	113.77	10.3
8	True	ε	7.9	5	106.91	10.36
9	True	ε	6.92	5	125.39	9.57
10	True	ε	7.83	7	130.59	10.8
11	True	ε	7.16	4	123.11	8.81
12	True	ε	7.43	6	139.79	10.4
13	True	ε	9.28	5	140.84	10.85
14	True	ε	8.39	5	129.4	9.51
15	True	ε	8.14	5	148.46	9.51
16	True	ε	8.37	5	139.87	8.78
17	True	ε	7.61	6	124.48	7.85
18	True	ε	7.26	5	113.92	11.62
19	True	ε	7.78	5	125.24	10.9
20	True	ε	10.02	5	128.47	9.02

A.4 GRASP parameter tuning tables

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.1	0.1	25	6	7.896	11.21	3.35	4.696	11.45
0.1	0.1	50	5.43	6.829	8.71	7.04	7.518	7.93
0.1	0.1	75	5.75	7.024	9.31	9.94	11.239	12.25
0.1	0.1	100	4.79	6.03	8.22	14.16	14.88	15.72
0.1	0.3	25	4.42	6.815	8.37	3.34	3.969	4.73
0.1	0.3	50	5.01	6.03	7.49	6.55	7.288	8.04
0.1	0.3	75	5.12	5.78	6.96	10.27	11.129	11.95
0.1	0.3	100	5.16	6.147	7.27	12.82	14.183	15.71
0.1	0.5	25	4.2	6.096	8.32	3.51	3.947	4.28
0.1	0.5	50	5.54	6.375	7.19	6.36	7.305	8.04
0.1	0.5	75	5.21	5.747	7.1	10.49	11.306	12.05
0.1	0.5	100	4.7	5.771	7.52	12.87	14.155	15.83
0.1	0.7	25	5.28	6.468	8.47	3.58	4.072	4.4
0.1	0.7	50	4.69	5.985	6.62	6.76	7.582	8.23
0.1	0.7	75	4.63	5.889	7.41	10.78	11.374	12.2
0.1	0.7	100	4.47	5.956	7.81	13.65	14.947	15.96
0.3	0.1	25	5.97	8.066	10.75	3.79	4.019	4.47
0.3	0.1	50	5.31	7.519	10.19	7.12	7.689	8.14
0.3	0.1	75	5.87	7.121	8.55	10.66	10.998	11.28
0.3	0.1	100	5.5	6.791	8.78	14.04	14.512	14.99
0.3	0.3	25	5.61	7.668	8.9	3.37	3.956	4.47
0.3	0.3	50	5.84	7.037	9.19	7.09	7.695	8.48

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.3	0.3	75	4.73	6.515	7.86	10.54	11.158	12.28
0.3	0.3	100	4.52	6.055	6.76	13.73	14.758	16.31
0.3	0.5	25	5.69	7.018	7.72	3.75	4.158	4.5
0.3	0.5	50	5.97	6.936	8.81	7.37	7.73	8.12
0.3	0.5	75	5.62	6.843	7.63	10.11	11.197	12.28
0.3	0.5	100	5.54	6.586	7.7	13.03	14.686	15.7
0.3	0.7	25	5.1	6.985	8.68	3.7	4.089	4.51
0.3	0.7	50	5.31	6.603	8.15	7.26	7.825	8.49
0.3	0.7	75	4.63	6.255	7.68	10.78	11.531	12.74
0.3	0.7	100	4.48	6.11	7.42	14.04	14.573	15.28
0.5	0.1	25	5.59	8.195	9.55	3.73	4.054	4.36
0.5	0.1	50	4.79	7.193	8.55	6.74	7.606	8.36
0.5	0.1	75	5.36	7.154	8.57	10.22	11.051	11.75
0.5	0.1	100	4.57	7.051	9.15	13.73	14.573	15.2
0.5	0.3	25	5.64	7.158	9.1	3.3	3.996	4.53
0.5	0.3	50	5.69	6.89	8.16	7	7.57	8.28
0.5	0.3	75	4.67	6.726	8.44	10.14	10.903	12.39
0.5	0.3	100	4.25	6.099	7.24	13.22	14.566	18.5
0.5	0.5	25	5.28	7.282	9.04	3.7	4.13	4.57
0.5	0.5	50	4.65	6.99	9	6.53	7.522	8.02
0.5	0.5	75	5.21	6.745	7.62	10.31	11.273	12.66
0.5	0.5	100	5.7	6.464	8.08	14	14.592	15.28
0.5	0.7	25	5.75	7.308	9.04	3.62	4.096	4.53
0.5	0.7	50	4.16	5.862	8.19	7.12	7.775	8.45
0.5	0.7	75	5.33	6.564	7.82	9.96	11.178	12.19

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.5	0.7	100	4.81	6.34	8.44	13.57	14.87	15.71
0.7	0.1	25	5.05	7.44	10.09	3.68	4.072	4.56
0.7	0.1	50	5.29	7.455	9.23	6.76	7.598	8.06
0.7	0.1	75	5.63	7.264	9.04	10.02	11.089	12.33
0.7	0.1	100	5.17	6.342	7.76	13.35	14.3	15.9
0.7	0.3	25	5.82	7.294	9.92	3.49	3.93	4.37
0.7	0.3	50	4.84	6.401	8.29	6.74	7.573	8.19
0.7	0.3	75	5.16	6.595	7.79	9.8	11.004	11.89
0.7	0.3	100	5.51	6.382	7.21	13.77	14.398	15.89
0.7	0.5	25	5.97	7.397	9.19	3.85	4.32	5.28
0.7	0.5	50	5.23	6.451	8.54	6.96	7.39	7.92
0.7	0.5	75	5.27	6.446	8.59	10.4	11.25	12.21
0.7	0.5	100	5.46	6.69	8.12	13.65	14.416	15.63
0.7	0.7	25	5.41	7.697	10.78	3.44	4.185	4.72
0.7	0.7	50	4.75	6.782	8.54	7.1	7.699	8.19
0.7	0.7	75	5.2	6.49	7.82	9.92	11.224	12.19
0.7	0.7	100	5.11	6.081	7.31	13.6	14.95	16.36

Table A.28: GRASP fine-tuning results for Size 1 instances

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.1	0.1	25	6.17	7.53	8.76	37.26	41.36	46.98
0.1	0.1	50	6.6	7.67	9.77	63.95	71.56	79.46
0.1	0.1	75	6.27	7.55	9.44	96.33	106.04	118.2
0.1	0.1	100	5.47	7.24	9.05	131.78	142.35	157.37
0.1	0.3	25	7.19	8.33	9.15	31.48	36.86	40.45
0.1	0.3	50	6.45	7.74	8.78	64.92	72.85	83.03
0.1	0.3	75	6.43	7.33	7.99	94.64	106.66	124.66
0.1	0.3	100	6.39	7.34	8.5	120.39	140.84	157.77
0.1	0.5	25	6.6	8.12	9.46	32.81	38.5	43.95
0.1	0.5	50	6.22	7.43	8.43	65.44	75.54	86.64
0.1	0.5	75	6.8	7.86	8.98	101.89	113.15	127.44
0.1	0.5	100	6.41	7.5	8.95	130.38	147.69	165.84
0.1	0.7	25	6.69	7.95	9.26	36.96	41.03	45.56
0.1	0.7	50	6.72	7.86	8.65	72.18	80.47	91.86
0.1	0.7	75	7.18	7.74	8.19	107.41	119.37	138.98
0.1	0.7	100	6.42	7.36	8.42	142.22	156.53	175.53
0.3	0.1	25	7.86	8.79	10.93	33.02	37.56	41.24
0.3	0.1	50	6.82	8.15	9.49	67.69	74.65	80.05
0.3	0.1	75	5.97	7.73	10.15	100.14	110.55	120.57
0.3	0.1	100	6.59	7.26	8.44	132.24	143.76	154.68
0.3	0.3	25	6.82	7.91	10.39	33.83	38.13	42.63
0.3	0.3	50	6.36	8.11	9.91	69.59	74.87	81.13
0.3	0.3	75	6.18	7.66	8.46	97.61	111.18	125.26
0.3	0.3	100	6.66	7.51	8.91	129.13	147.61	165.13

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.3	0.5	25	7.74	8.32	9.11	35.48	40.51	46.51
0.3	0.5	50	7.45	8.51	9.81	70.67	79.19	90.1
0.3	0.5	75	7.18	8.26	9.21	104.26	116.91	128.22
0.3	0.5	100	6.61	7.97	8.69	136.05	152.9	172.34
0.3	0.7	25	8.04	8.74	9.47	38.87	42.52	47.25
0.3	0.7	50	7.13	8.68	10.06	76.55	82.98	92.63
0.3	0.7	75	6.58	7.69	8.85	111.01	124.56	139.04
0.3	0.7	100	6.17	7.4	8.7	150.06	164.71	187.28
0.5	0.1	25	7.35	8.69	10.69	33.52	38.08	40.38
0.5	0.1	50	6.67	7.82	9.97	68.5	75.69	81.67
0.5	0.1	75	6.59	7.75	9.22	103.33	113.05	118.93
0.5	0.1	100	5.74	7.58	9.86	135.77	147.85	155.75
0.5	0.3	25	6.77	8.3	9.29	36.01	38.93	44.05
0.5	0.3	50	6.24	8.15	9.42	70.85	76.95	82.05
0.5	0.3	75	6.7	7.94	9.33	105.21	113.98	125.98
0.5	0.3	100	6.79	7.72	8.69	139.64	149.68	165.18
0.5	0.5	25	7.57	8.73	9.76	36.87	40.75	45.75
0.5	0.5	50	6.75	8.14	9.43	74.11	80.22	87.49
0.5	0.5	75	7.37	7.94	9.4	110.98	119.89	131.17
0.5	0.5	100	5.87	7.8	8.69	145.64	157.69	178.4
0.5	0.7	25	7.4	8.65	9.85	37.52	43.5	49.58
0.5	0.7	50	4.88	7.66	8.87	78.07	86.77	97.81
0.5	0.7	75	6.6	8.02	9.21	117.06	127.2	144.39
0.5	0.7	100	6.53	7.71	8.39	151.83	167.12	184.81
0.7	0.1	25	7.28	8.55	11.08	36.07	39.18	41.55

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.7	0.1	50	6.49	7.64	10.35	70.86	77.73	81.21
0.7	0.1	75	6.53	7.49	9.26	106.47	114.23	123.11
0.7	0.1	100	5.43	7.08	9.54	136.44	151.39	158.22
0.7	0.3	25	7.16	8.59	9.47	36.87	39.57	44.23
0.7	0.3	50	7.4	7.91	8.8	74.33	79.89	89.21
0.7	0.3	75	6.28	7.53	9.49	106.64	117.07	127.05
0.7	0.3	100	5.69	7.73	9.64	140.18	153.92	169.95
0.7	0.5	25	6.68	8.55	11.24	39.56	41.34	44.64
0.7	0.5	50	6.66	7.85	9.02	74.17	80.61	88.41
0.7	0.5	75	6.85	7.77	8.52	110.73	121.06	130.91
0.7	0.5	100	5.85	7.62	8.84	142.74	160.38	179.97
0.7	0.7	25	7.81	8.8	10.17	41.24	44.47	48.08
0.7	0.7	50	7.53	8.5	9.49	78.36	87.07	94.29
0.7	0.7	75	6.39	7.44	8.33	120.84	130.08	143.76
0.7	0.7	100	4.49	7.49	8.99	156.56	170.61	187.39

Table A.29: GRASP fine-tuning results for Size 2 instances

A.5 GRASP Statistical tests

A.5.1 Statistical testing of Thread Tuning for Size 1 instances

Threads	Mean Runtime (s)	Standard Deviation (s)
1	38.28	1.88
2	33.47	2.11
3	24.32	1.39
4	22.96	1.25
5	19.00	0.98
6	18.62	0.91
7	17.01	0.98
8	16.89	0.66
9	16.09	0.70
10	15.60	0.62
11	15.32	0.77
12	15.34	0.68
13	14.82	0.61
14	14.74	0.42
15	14.48	0.49
16	14.60	0.64

Table A.30: Thread performance comparison for Size 1 instances

Repeated Measures ANOVA Summary

Source	F Value	Num DF	Den DF	Pr $> F$
Threads	1695.6740	15.0000	285.0000	0.0000

Table A.31: Summary of Repeated Measures ANOVA

The table contains the following:

Source: Refers to the independent variable or factor being tested, which in this case is “Threads”, F Value: Represents the test statistic calculated by the ANOVA, num DF: Indicates the degrees of freedom for the numerator, which is the number of levels of the independent variable minus 1, den DF: Represents the degrees of freedom for the denominator, which is the total number of observations minus the number of levels of the independent variable, Pr $> F$: Denotes the p-value associated with the F statistic, indicating the statistical significance of the results.

The repeated measures ANOVA tests if there are statistically significant differences in the means of the same subjects under different conditions (in this case, runtime under different thread counts). The results show that there is a statistically significant difference in runtimes across different thread counts.

Following a significant ANOVA result, the Tukey’s HSD post-hoc test helps identify which specific group(s) differ from each other. Tukey’s HSD (Honestly Significant Difference) post-hoc test is a multiple comparison test used to find which means among a set of means differ from the rest. It is commonly used after a significant ANOVA result to determine which groups are different. The test compares all possible pairs of means and controls the familywise error rate.

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
1	2	-4809.85	0.0	-5962.0556	-3657.6444	True
1	3	-13961.85	0.0	-15114.0556	-12809.6444	True
1	4	-15317.35	0.0	-16469.5556	-14165.1444	True
1	5	-19279.9	0.0	-20432.1056	-18127.6944	True
1	6	-19662.85	0.0	-20815.0556	-18510.6444	True
1	7	-21270.95	0.0	-22423.1556	-20118.7444	True
1	8	-21387.25	0.0	-22539.4556	-20235.0444	True
1	9	-22187.1	0.0	-23339.3056	-21034.8944	True
1	10	-22683.45	0.0	-23835.6556	-21531.2444	True
1	11	-22960.0	0.0	-24112.2056	-21807.7944	True
1	12	-22937.5	0.0	-24089.7056	-21785.2944	True
1	13	-23461.05	0.0	-24613.2556	-22308.8444	True
1	14	-23545.4	0.0	-24697.6056	-22393.1944	True
1	15	-23803.15	0.0	-24955.3556	-22650.9444	True
1	16	-23681.65	0.0	-24833.8556	-22529.4444	True
2	3	-9152.0	0.0	-10304.2056	-7999.7944	True
2	4	-10507.5	0.0	-11659.7056	-9355.2944	True
2	5	-14470.05	0.0	-15622.2556	-13317.8444	True
2	6	-14853.0	0.0	-16005.2056	-13700.7944	True
2	7	-16461.1	0.0	-17613.3056	-15308.8944	True
2	8	-16577.4	0.0	-17729.6056	-15425.1944	True
2	9	-17377.25	0.0	-18529.4556	-16225.0444	True
2	10	-17873.6	0.0	-19025.8056	-16721.3944	True
2	11	-18150.15	0.0	-19302.3556	-16997.9444	True
2	12	-18127.65	0.0	-19279.8556	-16975.4444	True
2	13	-18651.2	0.0	-19803.4056	-17498.9944	True
2	14	-18735.55	0.0	-19887.7556	-17583.3444	True
2	15	-18993.3	0.0	-20145.5056	-17841.0944	True
2	16	-18871.8	0.0	-20024.0056	-17719.5944	True
3	4	-1355.5	0.0061	-2507.7056	-203.2944	True
3	5	-5318.05	0.0	-6470.2556	-4165.8444	True
3	6	-5701.0	0.0	-6853.2056	-4548.7944	True
3	7	-7309.1	0.0	-8461.3056	-6156.8944	True
3	8	-7425.4	0.0	-8577.6056	-6273.1944	True
3	9	-8225.25	0.0	-9377.4556	-7073.0444	True
3	10	-8721.6	0.0	-9873.8056	-7569.3944	True
3	11	-8998.15	0.0	-10150.3556	-7845.9444	True
3	12	-8975.65	0.0	-10127.8556	-7823.4444	True
3	13	-9499.2	0.0	-10651.4056	-8346.9944	True
3	14	-9583.55	0.0	-10735.7556	-8431.3444	True
3	15	-9841.3	0.0	-10993.5056	-8689.0944	True
3	16	-9719.8	0.0	-10872.0056	-8567.5944	True

Table A.32: Tukey HSD test results for threads Size 1 Part 1

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
4	5	-3962.55	0.0	-5114.7556	-2810.3444	True
4	6	-4345.5	0.0	-5497.7056	-3193.2944	True
4	7	-5953.6	0.0	-7105.8056	-4801.3944	True
4	8	-6069.9	0.0	-7222.1056	-4917.6944	True
4	9	-6869.75	0.0	-8021.9556	-5717.5444	True
4	10	-7366.1	0.0	-8518.3056	-6213.8944	True
4	11	-7642.65	0.0	-8794.8556	-6490.4444	True
4	12	-7620.15	0.0	-8772.3556	-6467.9444	True
4	13	-8143.7	0.0	-9295.9056	-6991.4944	True
4	14	-8228.05	0.0	-9380.2556	-7075.8444	True
4	15	-8485.8	0.0	-9638.0056	-7333.5944	True
4	16	-8364.3	0.0	-9516.5056	-7212.0944	True
5	6	-382.95	0.9989	-1535.1556	769.2556	False
5	7	-1991.05	0.0	-3143.2556	-838.8444	True
5	8	-2107.35	0.0	-3259.5556	-955.1444	True
5	9	-2907.2	0.0	-4059.4056	-1754.9944	True
5	10	-3403.55	0.0	-4555.7556	-2251.3444	True
5	11	-3680.1	0.0	-4832.3056	-2527.8944	True
5	12	-3657.6	0.0	-4809.8056	-2505.3944	True
5	13	-4181.15	0.0	-5333.3556	-3028.9444	True
5	14	-4265.5	0.0	-5417.7056	-3113.2944	True
5	15	-4523.25	0.0	-5675.4556	-3371.0444	True
5	16	-4401.75	0.0	-5553.9556	-3249.5444	True
6	7	-1608.1	0.0003	-2760.3056	-455.8944	True
6	8	-1724.4	0.0	-2876.6056	-572.1944	True
6	9	-2524.25	0.0	-3676.4556	-1372.0444	True
6	10	-3020.6	0.0	-4172.8056	-1868.3944	True
6	11	-3297.15	0.0	-4449.3556	-2144.9444	True
6	12	-3274.65	0.0	-4426.8556	-2122.4444	True
6	13	-3798.2	0.0	-4950.4056	-2645.9944	True
6	14	-3882.55	0.0	-5034.7556	-2730.3444	True
6	15	-4140.3	0.0	-5292.5056	-2988.0944	True
6	16	-4018.8	0.0	-5171.0056	-2866.5944	True
7	8	-116.3	1.0	-1268.5056	1035.9056	False
7	9	-916.15	0.306	-2068.3556	236.0556	False
7	10	-1412.5	0.0031	-2564.7056	-260.2944	True
7	11	-1689.05	0.0001	-2841.2556	-536.8444	True
7	12	-1666.55	0.0001	-2818.7556	-514.3444	True
7	13	-2190.1	0.0	-3342.3056	-1037.8944	True
7	14	-2274.45	0.0	-3426.6556	-1122.2444	True
7	15	-2532.2	0.0	-3684.4056	-1379.9944	True
7	16	-2410.7	0.0	-3562.9056	-1258.4944	True

Table A.33: Tukey HSD test results for threads Size 1 Part 2

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
8	9	-799.85	0.5502	-1952.0556	352.3556	False
8	10	-1296.2	0.0117	-2448.4056	-143.9944	True
8	11	-1572.75	0.0004	-2724.9556	-420.5444	True
8	12	-1550.25	0.0005	-2702.4556	-398.0444	True
8	13	-2073.8	0.0	-3226.0056	-921.5944	True
8	14	-2158.15	0.0	-3310.3556	-1005.9444	True
8	15	-2415.9	0.0	-3568.1056	-1263.6944	True
8	16	-2294.4	0.0	-3446.6056	-1142.1944	True
9	10	-496.35	0.9832	-1648.5556	655.8556	False
9	11	-772.9	0.6109	-1925.1056	379.3056	False
9	12	-750.4	0.6606	-1902.6056	401.8056	False
9	13	-1273.95	0.0149	-2426.1556	-121.7444	True
9	14	-1358.3	0.0059	-2510.5056	-206.0944	True
9	15	-1616.05	0.0002	-2768.2556	-463.8444	True
9	16	-1494.55	0.0011	-2646.7556	-342.3444	True
10	11	-276.55	1.0	-1428.7556	875.6556	False
10	12	-254.05	1.0	-1406.2556	898.1556	False
10	13	-777.6	0.6003	-1929.8056	374.6056	False
10	14	-861.95	0.4133	-2014.1556	290.2556	False
10	15	-1119.7	0.0671	-2271.9056	32.5056	False
10	16	-998.2	0.1781	-2150.4056	154.0056	False
11	12	22.5	1.0	-1129.7056	1174.7056	False
11	13	-501.05	0.9816	-1653.2556	651.1556	False
11	14	-585.4	0.9297	-1737.6056	566.8056	False
11	15	-843.15	0.4537	-1995.3556	309.0556	False
11	16	-721.65	0.7214	-1873.8556	430.5556	False
12	13	-523.55	0.9725	-1675.7556	628.6556	False
12	14	-607.9	0.9064	-1760.1056	544.3056	False
12	15	-865.65	0.4055	-2017.8556	286.5556	False
12	16	-744.15	0.6741	-1896.3556	408.0556	False
13	14	-84.35	1.0	-1236.5556	1067.8556	False
13	15	-342.1	0.9997	-1494.3056	810.1056	False
13	16	-220.6	1.0	-1372.8056	931.6056	False
14	15	-257.75	1.0	-1409.9556	894.4556	False
14	16	-136.25	1.0	-1288.4556	1015.9556	False
15	16	121.5	1.0	-1030.7056	1273.7056	False

Table A.34: Tukey HSD test results for threads Size 1 Part 3

The results of the Tukey Honestly Significant Difference (HSD) test reveal significant differences in the mean values of the dependent variable across different groups (Groups 1 to 16). Specifically:

There is a statistically significant difference ($p < 0.05$) in the mean values of the dependent variable between Group 1 and all other groups (Groups 2 to 16). Significant differences exist among most pairs of groups, with a few exceptions: a) There is no statistically significant difference between Group 5 and Group 6 ($p = 0.9989$). b) Group 7 does not significantly differ from Group 8 ($p = 1.0$) or Group 9 ($p = 0.306$). c) Group 8 does not significantly differ from Group 9 ($p = 0.5502$). d) Groups 9, 10, 11, and 12 do not significantly differ from each other (all $p > 0.05$). e) Groups 10 through 16 show no significant differences among themselves (all $p > 0.05$). The largest mean differences are observed between Group 1 and the higher-numbered groups, with the difference increasing as the group number increases. From Group 10 onwards, the differences between consecutive groups become smaller and statistically insignificant.

These findings suggest that the number of threads significantly influences the mean values of the dependent variable, which in this case it is the runtime, with Group 1 showing significant differences compared to all other groups, while certain pairs of groups exhibit no discernible differences. These results provide valuable insights into the relationship between the number of threads and the dependent variable, contributing to a deeper understanding of the experimental conditions under investigation.

Even though on average, the runtime does go down as more threads are added, the improvement eventually plateaus.

Effect Sizes (Cohen's d compared to 1 thread): Effect size is a measure of the magnitude of the difference between groups. Here, Cohen's d is calculated to quantify the difference in runtimes between 1 thread and the other thread counts.

Comparison	**Cohen's d**
Group 1 vs. Group 2	2.40
Group 1 vs. Group 3	8.44
Group 1 vs. Group 4	9.58
Group 1 vs. Group 5	12.82
Group 1 vs. Group 6	13.29
Group 1 vs. Group 7	14.15
Group 1 vs. Group 8	15.16
Group 1 vs. Group 9	15.62
Group 1 vs. Group 10	16.17
Group 1 vs. Group 11	15.95
Group 1 vs. Group 12	16.19
Group 1 vs. Group 13	16.75
Group 1 vs. Group 14	17.25
Group 1 vs. Group 15	17.28
Group 1 vs. Group 16	16.82

Table A.35: Cohen's d effect sizes for pairwise comparisons

These effect sizes indicate large differences between the mean values of the dependent variable for Group 1 compared to each of the other groups. The magnitude of these differences, as measured by Cohen's d, is substantial, suggesting a strong effect of the number of threads on the dependent variable.

Paired t-tests between adjacent thread counts: The paired t-tests are used to compare the means of the same group at two different times (in this case, runtimes between adjacent thread counts). The results provide insights into the significance of the performance differences between consecutive thread counts.

Pair	T**-Value**	p**-value**
1 vs. 2	15.07	5.07×10^{-12}
2 vs. 3	26.32	2.06×10^{-16}
3 vs. 4	6.29	4.89×10^{-6}
4 vs. 5	18.68	1.09×10^{-13}
5 vs. 6	1.99	0.0616
6 vs. 7	10.32	3.17×10^{-9}
7 vs. 8	0.49	0.6311
8 vs. 9	4.46	2.66×10^{-4}
9 vs. 10	4.01	7.48×10^{-4}
10 vs. 11	2.21	0.0399
11 vs. 12	-0.16	0.8779
12 vs. 13	3.68	0.0016
13 vs. 14	0.70	0.4947
14 vs. 15	2.68	0.0147
15 vs. 16	-0.78	0.4439

Table A.36: Paired t-tests results

Our paired t-tests indicate significant differences in mean values of the dependent variable between several pairs of groups. Specifically:

The mean values of Group 1 and Group 2 differ significantly ($t = 15.07, p < 0.001$). Significant differences are also observed between Group 2 and Group 3 ($t = 26.32, p < 0.001$), Group 3 and Group 4 ($t = 6.29, p < 0.001$), and Group 4 and Group 5 ($t = 18.68, p < 0.001$). The difference between Group 5 and Group 6 is not statistically significant at the conventional 0.05 level, but it is close ($t = 1.99, p = 0.0616$). Significant differences re-emerge between Group 6 and Group 7 ($t = 10.32, p < 0.001$). No significant difference is observed between Group 7 and

Group 8 ($t = 0.49, p = 0.6311$). Significant differences are found again between Group 8 and Group 9 ($t = 4.46, p < 0.001$), and Group 9 and Group 10 ($t = 4.01, p < 0.001$). The difference between Group 10 and Group 11 is significant at the 0.05 level ($t = 2.21, p = 0.0399$). No significant difference is found between Group 11 and Group 12 ($t = -0.16, p = 0.8779$). Significant differences are observed between Group 12 and Group 13 ($t = 3.68, p = 0.0016$). No significant differences are found between Group 13 and Group 14 ($t = 0.70, p = 0.4947$). A significant difference is observed between Group 14 and Group 15 ($t = 2.68, p = 0.0147$). Finally, no significant difference is found between Group 15 and Group 16 ($t = -0.78, p = 0.4439!$).

These findings suggest nuanced variations in the dependent variable across different groups, highlighting specific pairs where the differences are statistically significant. The pattern of significance indicates that:

The most substantial changes occur in the earlier groups (1 through 5), with highly significant differences between each pair. As the group numbers increase (representing an increase in the number of threads), the differences between adjacent groups become less consistently significant. There is a pattern of alternating significance and non-significance in the later groups, suggesting that performance improvements may plateau or become more incremental with higher thread counts. The lack of significant difference between the last two groups (15 and 16) might indicate that increasing the thread count beyond a certain point may not yield substantial performance improvements.

This pattern supports the hypothesis that as the number of threads increases, the difference between the values becomes less statistically significant, particularly in the higher thread count ranges. This could be due to factors such as diminishing returns from parallelization, potential overhead from thread management, or hardware limitations.

The one-way ANOVA test on the impact of threads on objective function

resulted in a $F-$statistic of approximately 0.1565 and a p-value of approximately 0.999. The high p-value suggests that there is no significant difference in the objective function value across different numbers of threads, meaning the number of threads does not have a statistically significant impact on the value based on this dataset.

In conclusion, increasing the number of threads has a statistically significant impact on the runtime, with no impact on the objective function value. Increasing from 1 to 10 threads and upwards shows no statistical difference, so increases beyond 10 may have diminishing returns.

A.5.2 Statistical testing of Thread Tuning for Size 2 instances

Threads	Mean Runtime (s)	Standard Deviation (s)
1	365.47	25.97
2	292.14	18.58
3	188.49	12.20
4	170.20	11.38
5	129.59	7.57
6	123.61	7.20
7	106.63	5.36
8	103.74	5.71
9	93.61	5.02
10	87.72	5.73
11	85.97	5.23
12	85.81	6.15
13	77.93	4.07
14	79.09	4.64
15	80.94	4.38
16	84.00	5.54

Table A.37: Thread performance comparison for Size 2 instances

Source	F Value	Num DF	Den DF	Pr > F
Threads	3086.2645	15.0000	285.0000	0.0000

Table A.38: Summary of Repeated Measures ANOVA

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
1	2	-73329.25	0.0	-84524.2403	-62134.2597	True
1	3	-176978.8	0.0	-188173.7903	-165783.8097	True
1	4	-195266.15	0.0	-206461.1403	-184071.1597	True
1	5	-235877.25	0.0	-247072.2403	-224682.2597	True
1	6	-241857.8	0.0	-253052.7903	-230662.8097	True
1	7	-258834.5	0.0	-270029.4903	-247639.5097	True
1	8	-261731.7	0.0	-272926.6903	-250536.7097	True
1	9	-271862.85	0.0	-283057.8403	-260667.8597	True
1	10	-277751.75	0.0	-288946.7403	-266556.7597	True
1	11	-279494.1	0.0	-290689.0903	-268299.1097	True
1	12	-279663.2	0.0	-290858.1903	-268468.2097	True
1	13	-287537.25	0.0	-298732.2403	-276342.2597	True
1	14	-286376.55	0.0	-297571.5403	-275181.5597	True
1	15	-284531.5	0.0	-295726.4903	-273336.5097	True
1	16	-281465.25	0.0	-292660.2403	-270270.2597	True
2	3	-103649.55	0.0	-114844.5403	-92454.5597	True
2	4	-121936.9	0.0	-133131.8903	-110741.9097	True
2	5	-162548.0	0.0	-173742.9903	-151353.0097	True
2	6	-168528.55	0.0	-179723.5403	-157333.5597	True
2	7	-185505.25	0.0	-196700.2403	-174310.2597	True
2	8	-188402.45	0.0	-199597.4403	-177207.4597	True
2	9	-198533.6	0.0	-209728.5903	-187338.6097	True
2	10	-204422.5	0.0	-215617.4903	-193227.5097	True
2	11	-206164.85	0.0	-217359.8403	-194969.8597	True
2	12	-206333.95	0.0	-217528.9403	-195138.9597	True
2	13	-214208.0	0.0	-225402.9903	-203013.0097	True
2	14	-213047.3	0.0	-224242.2903	-201852.3097	True
2	15	-211202.25	0.0	-222397.2403	-200007.2597	True
2	16	-208136.0	0.0	-219330.9903	-196941.0097	True
3	4	-18287.35	0.0	-29482.3403	-7092.3597	True
3	5	-58898.45	0.0	-70093.4403	-47703.4597	True
3	6	-64879.0	0.0	-76073.9903	-53684.0097	True
3	7	-81855.7	0.0	-93050.6903	-70660.7097	True
3	8	-84752.9	0.0	-95947.8903	-73557.9097	True
3	9	-94884.05	0.0	-106079.0403	-83689.0597	True
3	10	-100772.95	0.0	-111967.9403	-89577.9597	True
3	11	-102515.3	0.0	-113710.2903	-91320.3097	True
3	12	-102684.4	0.0	-113879.3903	-91489.4097	True
3	13	-110558.45	0.0	-121753.4403	-99363.4597	True
3	14	-109397.75	0.0	-120592.7403	-98202.7597	True
3	15	-107552.7	0.0	-118747.6903	-96357.7097	True
3	16	-104486.45	0.0	-115681.4403	-93291.4597	True

Table A.39: Tukey HSD test results for threads Size 2 Part 1

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
4	5	-40611.1	0.0	-51806.0903	-29416.1097	True
4	6	-46591.65	0.0	-57786.6403	-35396.6597	True
4	7	-63568.35	0.0	-74763.3403	-52373.3597	True
4	8	-66465.55	0.0	-77660.5403	-55270.5597	True
4	9	-76596.7	0.0	-87791.6903	-65401.7097	True
4	10	-82485.6	0.0	-93680.5903	-71290.6097	True
4	11	-84227.95	0.0	-95422.9403	-73032.9597	True
4	12	-84397.05	0.0	-95592.0403	-73202.0597	True
4	13	-92271.1	0.0	-103466.0903	-81076.1097	True
4	14	-91110.4	0.0	-102305.3903	-79915.4097	True
4	15	-89265.35	0.0	-100460.3403	-78070.3597	True
4	16	-86199.1	0.0	-97394.0903	-75004.1097	True
5	6	-5980.55	0.8974	-17175.5403	5214.4403	False
5	7	-22957.25	0.0	-34152.2403	-11762.2597	True
5	8	-25854.45	0.0	-37049.4403	-14659.4597	True
5	9	-35985.6	0.0	-47180.5903	-24790.6097	True
5	10	-41874.5	0.0	-53069.4903	-30679.5097	True
5	11	-43616.85	0.0	-54811.8403	-32421.8597	True
5	12	-43785.95	0.0	-54980.9403	-32590.9597	True
5	13	-51660.0	0.0	-62854.9903	-40465.0097	True
5	14	-50499.3	0.0	-61694.2903	-39304.3097	True
5	15	-48654.25	0.0	-59849.2403	-37459.2597	True
5	16	-45588.0	0.0	-56782.9903	-34393.0097	True
6	7	-16976.7	0.0	-28171.6903	-5781.7097	True
6	8	-19873.9	0.0	-31068.8903	-8678.9097	True
6	9	-30005.05	0.0	-41200.0403	-18810.0597	True
6	10	-35893.95	0.0	-47088.9403	-24698.9597	True
6	11	-37636.3	0.0	-48831.2903	-26441.3097	True
6	12	-37805.4	0.0	-49000.3903	-26610.4097	True
6	13	-45679.45	0.0	-56874.4403	-34484.4597	True
6	14	-44518.75	0.0	-55713.7403	-33323.7597	True
6	15	-42673.7	0.0	-53868.6903	-31478.7097	True
6	16	-39607.45	0.0	-50802.4403	-28412.4597	True
7	8	-2897.2	0.9999	-14092.1903	8297.7903	False
7	9	-13028.35	0.0072	-24223.3403	-1833.3597	True
7	10	-18917.25	0.0	-30112.2403	-7722.2597	True
7	11	-20659.6	0.0	-31854.5903	-9464.6097	True
7	12	-20828.7	0.0	-32023.6903	-9633.7097	True
7	13	-28702.75	0.0	-39897.7403	-17507.7597	True
7	14	-27542.05	0.0	-38737.0403	-16347.0597	True
7	15	-25697.0	0.0	-36891.9903	-14502.0097	True
7	16	-22630.75	0.0	-33825.7403	-11435.7597	True

Table A.40: Tukey HSD test results for threads Size 2 Part 2

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
8	9	-10131.15	0.1274	-21326.1403	1063.8403	False
8	10	-16020.05	0.0001	-27215.0403	-4825.0597	True
8	11	-17762.4	0.0	-28957.3903	-6567.4097	True
8	12	-17931.5	0.0	-29126.4903	-6736.5097	True
8	13	-25805.55	0.0	-37000.5403	-14610.5597	True
8	14	-24644.85	0.0	-35839.8403	-13449.8597	True
8	15	-22799.8	0.0	-33994.7903	-11604.8097	True
8	16	-19733.55	0.0	-30928.5403	-8538.5597	True
9	10	-5888.9	0.9084	-17083.8903	5306.0903	False
9	11	-7631.25	0.5828	-18826.2403	3563.7403	False
9	12	-7800.35	0.5435	-18995.3403	3394.6403	False
9	13	-15674.4	0.0002	-26869.3903	-4479.4097	True
9	14	-14513.7	0.0011	-25708.6903	-3318.7097	True
9	15	-12668.65	0.0108	-23863.6403	-1473.6597	True
9	16	-9602.4	0.1911	-20797.3903	1592.5903	False
10	11	-1742.35	1.0	-12937.3403	9452.6403	False
10	12	-1911.45	1.0	-13106.4403	9283.5403	False
10	13	-9785.5	0.1669	-20980.4903	1409.4903	False
10	14	-8624.8	0.3604	-19819.7903	2570.1903	False
10	15	-6779.75	0.7686	-17974.7403	4415.2403	False
10	16	-3713.5	0.9989	-14908.4903	7481.4903	False
11	12	-169.1	1.0	-11364.0903	11025.8903	False
11	13	-8043.15	0.4875	-19238.1403	3151.8403	False
11	14	-6882.45	0.7482	-18077.4403	4312.5403	False
11	15	-5037.4	0.9748	-16232.3903	6157.5903	False
11	16	-1971.15	1.0	-13166.1403	9223.8403	False
12	13	-7874.05	0.5264	-19069.0403	3320.9403	False
12	14	-6713.35	0.7814	-17908.3403	4481.6403	False
12	15	-4868.3	0.9816	-16063.2903	6326.6903	False
12	16	-1802.05	1.0	-12997.0403	9392.9403	False
13	14	1160.7	1.0	-10034.2903	12355.6903	False
13	15	3005.75	0.9999	-8189.2403	14200.7403	False
13	16	6072.0	0.8857	-5122.9903	17266.9903	False
14	15	1845.05	1.0	-9349.9403	13040.0403	False
14	16	4911.3	0.98	-6283.6903	16106.2903	False
15	16	3066.25	0.9999	-8128.7403	14261.2403	False

Table A.41: Tukey HSD test results for threads Size 2 Part 3

The results of the Tukey Honestly Significant Difference (HSD) test reveal significant differences in the mean runtime values across different thread groups (Groups 1 to 16). Specifically:

There is a statistically significant difference ($p < 0.05$) in the mean runtime between Group 1 (single thread) and all other groups (Groups 2 to 16). Significant differences exist among most pairs of groups, with some notable exceptions: a) There is no statistically significant difference between Group 5 and Group 6 ($p = 0.8974$). b) Group 7 does not significantly differ from Group 8 ($p = 0.9999$). c) Group 8 does not significantly differ from Group 9 ($p = 0.1274$). d) Groups 9, 10, 11, and 12 do not significantly differ from each other (all $p > 0.05$). e) Groups 10 through 16 show no significant differences among themselves (all $p > 0.05$). The largest mean differences are observed between Group 1 and the higher-numbered groups, with the difference increasing as the group number increases. The maximum mean difference is observed between Group 1 and Group 13 (-287,537.25). From Group 9 onwards, the differences between consecutive groups become smaller and statistically insignificant in most cases. The improvement in runtime is most pronounced in the early increases in thread count (from 1 to 8 threads), with statistically significant improvements observed at each step. After 9 threads, the improvements in runtime become less substantial and often not statistically significant, indicating a plateau in performance gains.

These findings suggest that the number of threads significantly influences the mean runtime, with single-threaded execution (Group 1) showing significant differences compared to all multi-threaded executions. The most substantial and statistically significant improvements occur when increasing threads from 1 to 9. Beyond 9 threads, while there are still some numerical improvements in runtime, these differences are often not statistically significant. This pattern indicates that there is a point of diminishing returns in adding more threads, likely due to factors such as increased overhead in thread management or limitations in the underlying hardware or workload parallelizability. The optimal thread count for this particular workload

appears to be around 9-13 threads, as further increases do not yield statistically significant improvements in runtime.

Cohen's d effect sizes:

Comparison	**Cohen's** d
Group 1 vs. Group 2	3.25
Group 1 vs. Group 3	8.72
Group 1 vs. Group 4	9.74
Group 1 vs. Group 5	12.33
Group 1 vs. Group 6	12.69
Group 1 vs. Group 7	13.80
Group 1 vs. Group 8	13.92
Group 1 vs. Group 9	14.53
Group 1 vs. Group 10	14.77
Group 1 vs. Group 11	14.92
Group 1 vs. Group 12	14.82
Group 1 vs. Group 13	15.47
Group 1 vs. Group 14	15.35
Group 1 vs. Group 15	15.28
Group 1 vs. Group 16	14.99

Table A.42: Cohen's d effect sizes for pairwise comparisons for Size 2

We get similar results as before, with the strongest effects when compared to running the single-threaded GRASP peaking around 13-14 threads.

Pair	**T-Value**	**p-value**
1 vs. 2	24.36	8.60×10^{-16}
2 vs. 3	44.61	1.07×10^{-20}
3 vs. 4	15.08	5.02×10^{-12}
4 vs. 5	33.05	2.97×10^{-18}
5 vs. 6	7.82	2.35×10^{-7}
6 vs. 7	17.68	2.97×10^{-13}
7 vs. 8	4.17	5.20×10^{-4}
8 vs. 9	18.67	1.11×10^{-13}
9 vs. 10	12.46	1.37×10^{-10}
10 vs. 11	2.96	8.12×10^{-3}
11 vs. 12	0.33	0.7447
12 vs. 13	12.16	2.07×10^{-10}
13 vs. 14	-2.68	0.0150
14 vs. 15	-3.75	1.36×10^{-3}
15 vs. 16	-3.29	3.89×10^{-3}

Table A.43: Paired t-tests results for Size 2

All pairs up to and including 9 vs. 10 show highly significant differences ($p < 0.001$), indicating substantial performance improvements with each increase in thread count up to 10 threads. The pair 10 vs. 11 shows a significant difference ($p < 0.01$), suggesting that there is still a measurable improvement when moving from 10 to 11 threads. The pair 11 vs. 12 shows no significant difference ($p = 0.7447$), indicating that the performance improvement plateaus at this point. Interestingly, the pair 12 vs. 13 shows a highly significant difference again ($p < 0.001$), suggesting a performance jump when moving to 13 threads. For pairs 13 vs. 14, 14 vs. 15, and 15 vs. 16, we see significant differences ($p < 0.05$), but with negative t-values. This suggests that performance actually decreases slightly with these additional threads.

This analysis suggests that the optimal thread count for this particular workload is likely 13-14, as it shows the last significant performance improvement before we start to see decreases. The slight performance decreases beyond 14 threads might be due to increased overhead from managing additional threads or resource contention.

The one-way ANOVA test on the impact of threads on objective function resulted in a $F-$statistic of approximately 0.1326 and a p-value of approximately 0.999. The high p-value suggests that there is no significant difference in the objective function value across different numbers of threads, meaning the number of threads does not have a statistically significant impact on the value based on this dataset, repeating the expected results from Size 1, so it is safe to say that using more threads has no impact, positive or negative, on the quality of the solution.

In conclusion, increasing the number of threads has a statistically significant impact on the runtime, with no impact on the objective function value. Increasing from 1 to 9 threads and upwards shows no statistical difference, and increases above 14 threads actually have negative performance, so 14 threads should be the ideal number to be used.

A.5.3 Statistical testing of Parameter Tuning for Size 1 instances

The One-Way ANOVA for the runtime of the GRASP Metaheuristic across different levels of iterations yielded an F-statistic of 8226.66 and a p-value of 0.0, indicating statistically significant differences in runtime across the iteration levels. For the α_a and α_l parameters, the runtime analysis resulted in F-statistics of 0.01 and 0.10, with p-values of 0.9985 and 0.9580 respectively, suggesting no significant impact on runtime. The analysis for the objective function value resulted in an F-statistic of 11.42 and a p-value of 2.652×10^{-7} for iterations, 4.83 and 0.0025 for α_a, and 7.09 and 1.089×10^{-4} for α_l, all showing statistically significant differences in solution

quality across these parameter levels.

Factorial ANOVA extends the concept of One-Way ANOVA by allowing for the examination of the effects of two or more independent variables (factors) simultaneously on a single dependent variable. It enables the analysis of not only the main effects of each factor but also the interaction effects between them. In the context of analyzing the performance of a metaheuristic algorithm, Factorial ANOVA allows us to understand not only how individual parameters (e.g., α_a, α_l and iterations) affect the algorithm's efficiency and solution quality but also how these parameters interact with each other to influence the outcomes. This comprehensive analysis is crucial for identifying optimal parameter settings, especially in complex systems where the interaction between parameters can significantly impact performance.

Effect	**Metric**	**F-Statistic**	**p-Value**
Time			
Main Effects - iters	Time	8347.26	< 0.000000
Main Effects - α_a	Time	0.43	0.735007
Main Effects - α_l	Time	4.18	0.006107
Interaction Effects (iters, α_a)	Time	0.85	0.572186
Interaction Effects (iters, α_l)	Time	0.94	0.493059
Interaction Effects (α_a, α_l)	Time	1.39	0.191076
Objective Value			
Main Effects - iters	Objective Value	11.64	< 0.0000002
Main Effects - α_a	Objective Value	5.07	0.001792
Main Effects - α_l	Objective Value	7.37	0.000075
Interaction Effects (iters, α_a)	Objective Value	0.14	0.998433
Interaction Effects (iters, α_l)	Objective Value	0.40	0.934506
Interaction Effects (α_a, α_l)	Objective Value	0.34	0.959751

Table A.44: Factorial ANOVA results for time and objective value for Size 1

The results show that the number of iterations had a very significant effect ($p < 0.000000$) on time, while α_a and α_l are not not significant. All interactions between parameters were not statistically significant. On the other hand, the number of iterations and both α parameters had a significant impact on objective values.

Detailed pairwise comparisons were conducted for both Time and Objective Function Value across different levels of iterations and α parameters identifying specific pairs that differ significantly. The Tukey HSD test results offer a comprehensive look at pairwise differences among the levels of iterations, highlighting significant disparities that inform the optimization of algorithm parameters.

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	3479.96	0.0	3298.29	3661.62	True
25	75	7076.15	0.0	6894.48	7257.82	True
25	100	10479.74	0.0	10298.08	10661.41	True
50	75	3596.19	0.0	3414.53	3777.86	True
50	100	6999.79	0.0	6818.12	7181.45	True
75	100	3403.59	0.0	3221.93	3585.26	True

Table A.45: Tukey HSD test results for iterations on runtime for Size 1

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	-3178.83	0.0045	-5611.92	-745.75	True
25	75	-3916.19	0.0002	-6349.28	-1483.11	True
25	100	-5336.96	0.0	-7770.04	-2903.87	True
50	75	-737.36	0.8632	-3170.45	1695.72	False
50	100	-2158.13	0.1026	-4591.21	274.96	False
75	100	-1420.76	0.4356	-3853.85	1012.32	False

Table A.46: Tukey HSD test results for iterations on objective value for Size 1

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	61.50	0.9991	-1084.63	1207.63	False
0.1	0.5	10.82	1.0	-1135.31	1156.95	False
0.1	0.7	-11.87	1.0	-1158.00	1134.26	False
0.3	0.5	-50.68	0.9995	-1196.81	1095.45	False
0.3	0.7	-73.37	0.9984	-1219.50	1072.76	False
0.5	0.7	-22.69	1.0	-1168.82	1123.44	False

Table A.47: Tukey HSD test results for α_a on runtime

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	3074.41	0.0077	604.60	5544.23	True
0.1	0.5	3046.51	0.0085	576.69	5516.32	True
0.1	0.7	2779.45	0.0202	309.63	5249.27	True
0.3	0.5	-27.91	1.0	-2497.72	2441.91	False
0.3	0.7	-294.96	0.9899	-2764.78	2174.85	False
0.5	0.7	-267.06	0.9925	-2736.87	2202.76	False

Table A.48: Tukey HSD test results for α_a on objective value for Size 1

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	-113.53	0.9942	-1259.40	1032.35	False
0.1	0.5	-32.23	0.9999	-1178.10	1113.65	False
0.1	0.7	129.78	0.9914	-1016.10	1275.65	False
0.3	0.5	81.30	0.9978	-1064.58	1227.18	False
0.3	0.7	243.30	0.9474	-902.58	1389.18	False
0.5	0.7	162.00	0.9835	-983.88	1307.88	False

Table A.49: Tukey HSD test results for α_l on runtime for Size 1

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	-3296.43	0.0033	-5753.47	-839.39	True
0.1	0.5	-3220.80	0.0043	-5677.84	-763.76	True
0.1	0.7	-4030.83	0.0002	-6487.86	-1573.79	True
0.3	0.5	75.63	0.9998	-2381.41	2532.67	False
0.3	0.7	-734.39	0.8680	-3191.43	1722.64	False
0.5	0.7	-810.03	0.8308	-3267.06	1647.01	False

Table A.50: Tukey HSD test results for α_l on objective value for Size 1

For the runtime of the algorithm, the test results show significant differences between all pairs of groups (p-adj = 0.0 for all comparisons). This indicates that the mean time is significantly different between each pair of groups tested. The magnitude of the mean difference (meandiff) increases as the difference between group numbers increases. For example, the difference between groups of 25 iterations and 50 iterations is smaller than the difference between groups of 25 iterations and 100 iterations. This suggests a trend where groups with higher iterations have significantly longer times. All comparisons were found to be significant, indicating that as the group number increases, so does the time, and these differences are not due to random chance.

Unlike the time, not all group comparisons for the objective function value are significant. Significant differences are observed in some pairs (e.g., 25 vs 50, 25 vs 75, 25 vs 100), while others show no significant difference. The significant differences do not follow a clear linear trend as observed with the time. This implies that the objective function value measure does not consistently increase or decrease with the group numbers in a manner that is statistically significant across all comparisons. The significant negative meandiff values (e.g., 25 vs 100, -5336.96) suggest that certain groups have significantly lower values compared to others. Specifically, group 25 tends to have higher value scores compared to other groups, indicating a decrease in value scores as the group number increases. However, when comparing 75 vs 100

iterations, the decrease is no longer statistically significant.

The time shows a clear and significant increase in values as the group number increases, suggesting a strong and consistent difference across all group comparisons.

The value, however, presents a more complex picture with significant differences only in certain group comparisons, indicating that while there are differences in value scores between specific groups, these differences are not uniformly observed across all groups.

α_a and α_l have no significant effect whatsoever on runtime, for any value they may take. Significant differences are observed between $\alpha_a = 0.1$ and all other values (0.3, 0.5, and 0.7). These comparisons have p-values < 0.05 and are rejecting the null hypothesis. The mean differences are positive, indicating that $\alpha_a = 0.1$ yields higher (worse) objective values compared to the other levels. There are no significant differences among values of 0.3, 0.5, and 0.7. Significant differences are observed between $\alpha_l = 0.1$ and all other values (0.3, 0.5, and 0.7). These comparisons have p-values < 0.05 and again reject the null hypothesis. The mean differences are negative, indicating that $\alpha_l = 0.1$ yields lower (better) objective values compared to the other levels. There are no significant differences among values of 0.3, 0.5, and 0.7.

These results suggest that while neither α_a nor α_l significantly affect runtime, they do impact the solution quality. Specifically, $\alpha_a = 0.1$ leads to worse solutions, while $\alpha_l = 0.1$ leads to better solutions compared to their respective higher values. For both parameters, values of 0.3, 0.5, and 0.7 perform similarly to each other in terms of solution quality.

Therefore, for instances of Size 1, the tuned parameters are set to the following values: iterations set to 100, although they increase time, they also improve objective function value (marginally when set to 100 iterations), but the time penalty is negligible; α_a set to 0.3 and α_l set to 0.1.

A.5.4 Statistical testing of Parameter Tuning for Size 2 instances

The One-Way ANOVA for the runtime of the GRASP Metaheuristic across different levels of iterations yielded an F-statistic of 5219.33 and a p-value of 0.0, indicating statistically significant differences in runtime across the iteration levels. For the α_a and α_l parameters, the runtime analysis resulted in F-statistics of 0.86 and 2.38, with p-values of 0.4637 and 0.0683 respectively, suggesting no significant impact on runtime. The analysis for the objective function value resulted in an F-statistic of 6.81 and a p-value of 1.602×10^{-4} for iterations, 1.46 and 0.2233 for α_a, and 0.51 and 0.6765 for α_l, showing statistically significant differences in solution quality across iteration levels, but not for the α parameters.

Effect	**Metric**	**F-Statistic**	**p-Value**
Time			
Main Effects - iters	Time	9562.34	< 0.000000
Main Effects - α_a	Time	40.01	1.364×10^{-23}
Main Effects - α_l	Time	110.64	4.445×10^{-57}
Interaction Effects (iters, α_a)	Time	3.76	1.281×10^{-4}
Interaction Effects (iters, α_l)	Time	7.91	4.337×10^{-11}
Interaction Effects (α_a, α_l)	Time	0.58	0.811264
Objective Value			
Main Effects - iters	Objective Value	6.58	0.000223
Main Effects - α_a	Objective Value	1.45	0.227379
Main Effects - α_l	Objective Value	0.51	0.678434
Interaction Effects (iters, α_a)	Objective Value	0.33	0.963511
Interaction Effects (iters, α_l)	Objective Value	0.15	0.998117
Interaction Effects (α_a, α_l)	Objective Value	0.12	0.999104

Table A.51: Factorial ANOVA results for time and objective value for Size 2

The results show that the number of iterations had a very significant effect ($p < 0.000000$) on time, while α_a and α_l also showed significant effects, contrary to the one-way ANOVA results. Interactions between iterations and both α parameters were statistically significant for time, but not between the α parameters themselves. For objective function values, only the number of iterations showed a significant impact, while neither the α parameters nor any interactions were significant. Detailed pairwise comparisons were conducted for both Time and Objective Function Value across different levels of iterations and α parameters identifying specific pairs that differ significantly. The Tukey HSD test results offer a comprehensive look at pairwise differences among the levels of iterations, highlighting significant disparities that inform the optimization of algorithm parameters.

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	38422.49	0.0	35962.50	40882.47	True
25	75	76418.65	0.0	73958.66	78878.64	True
25	100	113297.45	0.0	110837.46	115757.44	True
50	75	37996.16	0.0	35536.18	40456.15	True
50	100	74874.96	0.0	72414.98	77334.95	True
75	100	36878.80	0.0	34418.81	39338.79	True

Table A.52: Tukey HSD test results for iterations on runtime for Size 2

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	-4828.01	0.1775	-10949.99	1293.98	False
25	75	-7791.22	0.0061	-13913.20	-1669.24	True
25	100	-10179.63	0.0001	-16301.61	-4057.64	True
50	75	-2963.21	0.5973	-9085.20	3158.77	False
50	100	-5351.62	0.1107	-11473.60	770.36	False
75	100	-2388.41	0.7466	-8510.39	3733.58	False

Table A.53: Tukey HSD test results for iterations on objective value for Size 2

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	3237.18	0.908	-9189.15	15663.51	False
0.1	0.5	5409.20	0.6765	-7017.13	17835.53	False
0.1	0.7	7363.28	0.4223	-5063.05	19789.61	False
0.3	0.5	2172.02	0.9696	-10254.31	14598.35	False
0.3	0.7	4126.10	0.8278	-8300.23	16552.43	False
0.5	0.7	1954.08	0.9775	-10472.25	14380.41	False

Table A.54: Tukey HSD test results for α_a on runtime for Size 2

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	4385.91	0.2635	-1812.25	10584.06	False
0.1	0.5	4347.86	0.271	-1850.29	10546.02	False
0.1	0.7	2908.76	0.6215	-3289.40	9106.91	False
0.3	0.5	-38.04	1.0	-6236.20	6160.11	False
0.3	0.7	-1477.15	0.9277	-7675.30	4721.00	False
0.5	0.7	-1439.11	0.9326	-7637.26	4759.05	False

Table A.55: Tukey HSD test results for α_a on objective value for Size 2

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	872.45	0.9979	-11509.53	13254.43	False
0.1	0.5	5081.68	0.7157	-7300.30	17463.66	False
0.1	0.7	11496.06	0.0797	-885.92	23878.03	False
0.3	0.5	4209.23	0.8175	-8172.75	16591.21	False
0.3	0.7	10623.61	0.1216	-1758.37	23005.58	False
0.5	0.7	6414.38	0.5413	-5967.60	18796.35	False

Table A.56: Tukey HSD test results for α_l on runtime for Size 2

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	790.05	0.9879	-5422.02	7002.12	False
0.1	0.5	2650.46	0.6903	-3561.61	8862.53	False
0.1	0.7	2135.49	0.8124	-4076.58	8347.56	False
0.3	0.5	1860.41	0.8674	-4351.66	8072.48	False
0.3	0.7	1345.44	0.9444	-4866.63	7557.51	False
0.5	0.7	-514.98	0.9966	-6727.04	5697.09	False

Table A.57: Tukey HSD test results for α_l on objective value for Size 2

For the runtime of the algorithm, the test results show significant differences between all pairs of groups (p-adj = 0.0 for all comparisons). This indicates that the

mean time is significantly different between each pair of groups tested. The magnitude of the mean difference (meandiff) increases as the difference between group numbers increases. For example, the difference between groups of 25 iterations and 50 iterations is smaller than the difference between groups of 25 iterations and 100 iterations. This suggests a trend where groups with higher iterations have significantly longer times. All comparisons were found to be significant, indicating that as the group number increases, so does the time, and these differences are not due to random chance.

For the objective function value, significant differences are observed only in some pairs (25 vs 75, 25 vs 100), while others show no significant difference. The significant differences do not follow a clear linear trend as observed with the time. This implies that the objective function value measure does not consistently increase or decrease with the group numbers in a manner that is statistically significant across all comparisons. The significant negative meandiff values (e.g., 25 vs 100, -10179.63) suggest that certain groups have significantly lower values compared to others. Specifically, group 25 tends to have higher value scores compared to groups 75 and 100, indicating a decrease in value scores as the group number increases from 25 to 75 or 100. However, when comparing 50 vs 75, 50 vs 100, or 75 vs 100 iterations, the decreases are no longer statistically significant.

The time shows a clear and significant increase in values as the group number increases, suggesting a strong and consistent difference across all group comparisons. The value, however, presents a more complex picture with significant differences only in certain group comparisons, indicating that while there are differences in value scores between specific groups, these differences are not uniformly observed across all groups. Unlike in the previous results, α_a and α_l show no significant effect on either runtime or objective function value for any value they may take. All comparisons for both parameters across all levels have p-values > 0.05, failing to reject the null hypothesis. This suggests that changes in α_a and α_l do not lead to statistically significant changes in either runtime or solution quality.

These results suggest that while the number of iterations significantly affects both runtime and solution quality (to a certain extent), neither α_a nor α_l have a statistically significant impact on these metrics. This is in contrast to the factorial ANOVA results, which showed significant main effects for α_a and α_l on runtime. This discrepancy might be due to the different nature of the tests or potential interactions that are captured in the factorial ANOVA but not in the pairwise comparisons.

Therefore, for these instances of Size 2, the tuned parameters could be set as follows: iterations set to 75 or 100, as they improve objective function value significantly compared to 25 iterations, although they also increase time. The choice between 75 and 100 iterations would depend on the specific trade-off between solution quality and runtime that is acceptable for the problem at hand. As for α_a and α_l, given that they do not show statistically significant effects on either runtime or objective function value, their values could be set based on other considerations or left at default values, repeating the values for Size 1 of 0.3 and 0.1.

It is important to note that while the statistical tests do not show significant differences for α_a and α_l, there might still be practical differences that could be relevant in specific contexts. Additionally, the interaction effects observed in the factorial ANOVA for runtime suggest that the impact of these parameters might be more complex and intertwined with the number of iterations than the pairwise comparisons can reveal.

Future work might involve more detailed analysis of these interactions, or exploration of a wider range of values for α_a and α_l to see if significant effects emerge under different conditions. It might also be valuable to consider other performance metrics or to analyze the algorithm's behavior on different types of problem instances to get a more comprehensive understanding of the parameters' impacts.

A.6 Algorithms' graphical examples

A.6.1 The p-dispersion algorithm

Let us assume an instance of the problem with the configuration shown in the plot, with $|K| = 3$ to make the problem easier to follow. We have 3 center types along with their lower and upper bounds of centers to be used, figures and colors representing each type, along with 15 BUs, 14 centers and $p = 9$:

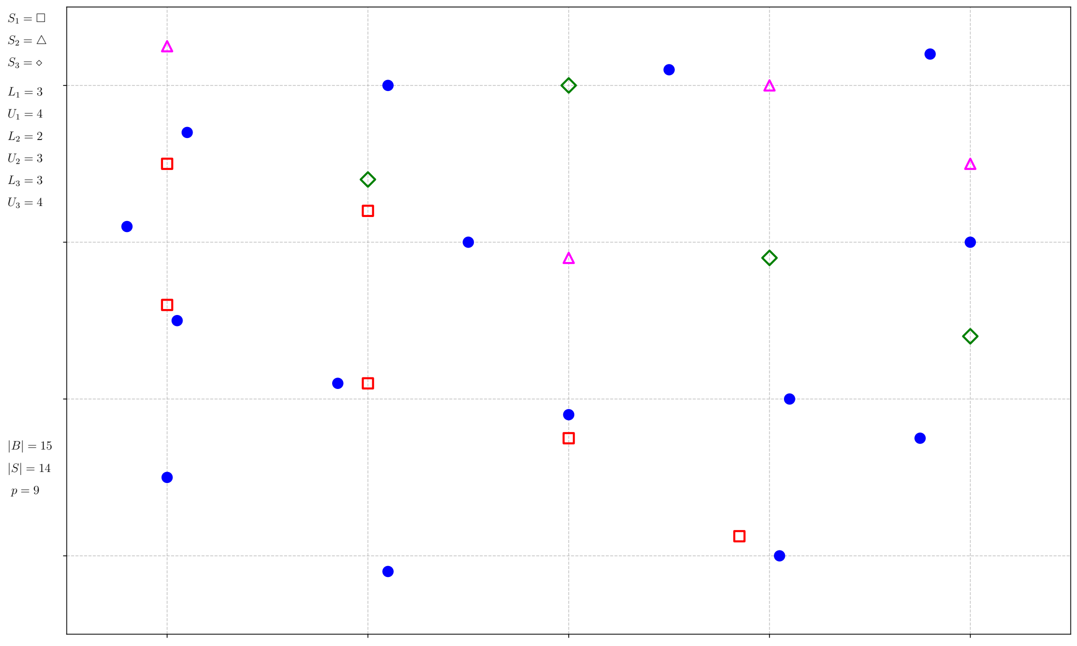


Figure A.1: Illustrative instance of the problem for p-dispersion algorithm

The location algorithm will solve $|K|$ p-dispersion subproblems with the heuristic presented in Algorithm 2. In this case, it will solve 3 subproblems, with the p of each problem set to the lower bound of the center type that it is trying to solve for, taking into account only the centers of that type, as follows:

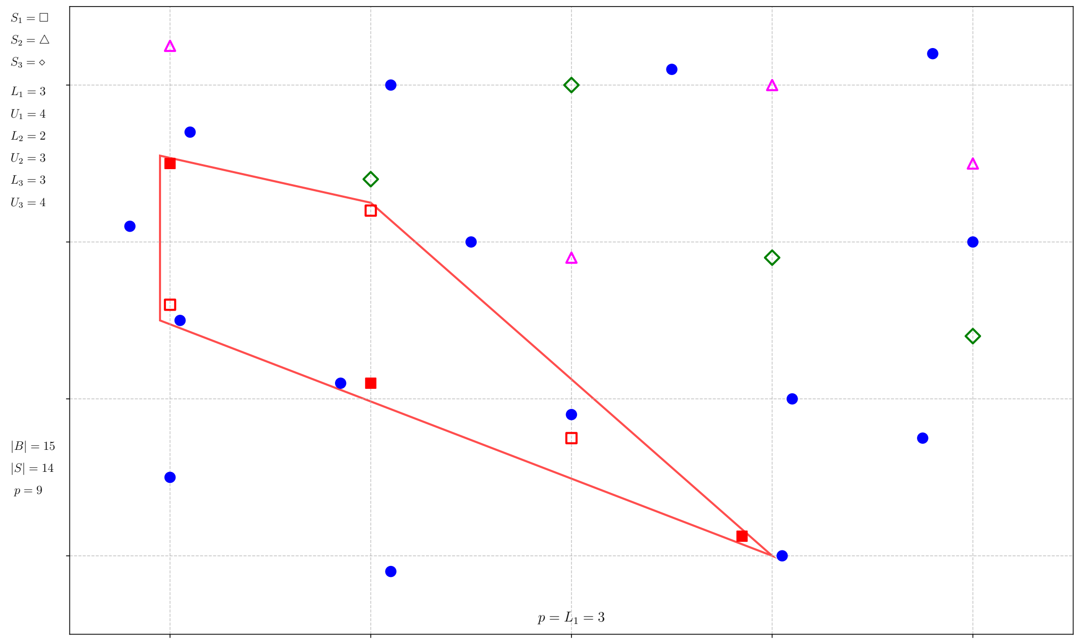


Figure A.2: Illustrative instance of the problem for p-dispersion for $k = 1$

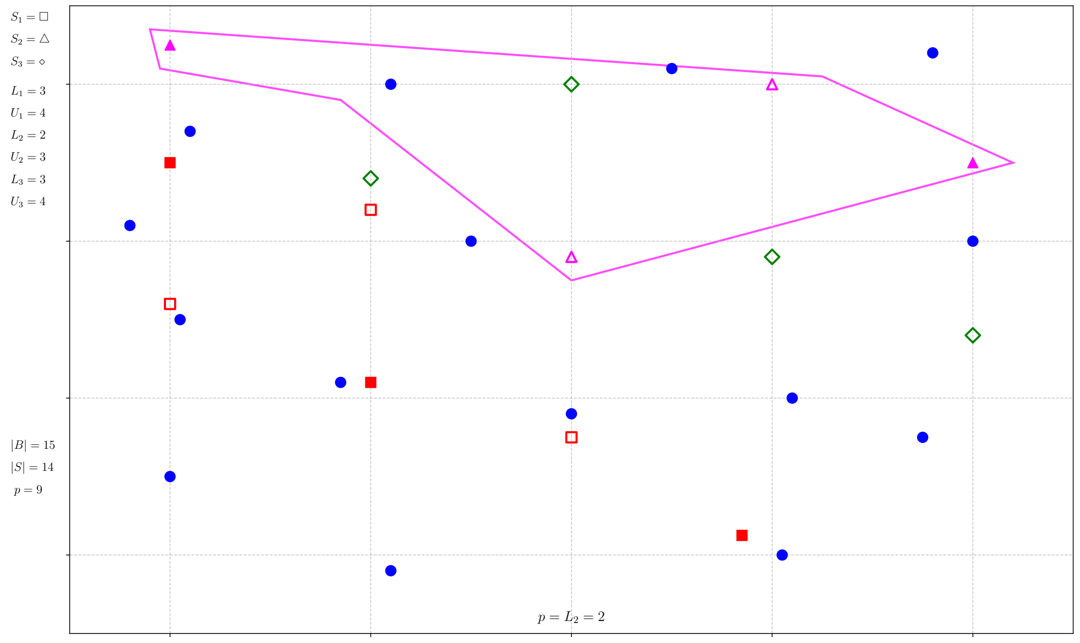


Figure A.3: Illustrative instance of the problem for p-dispersion for $k = 2$

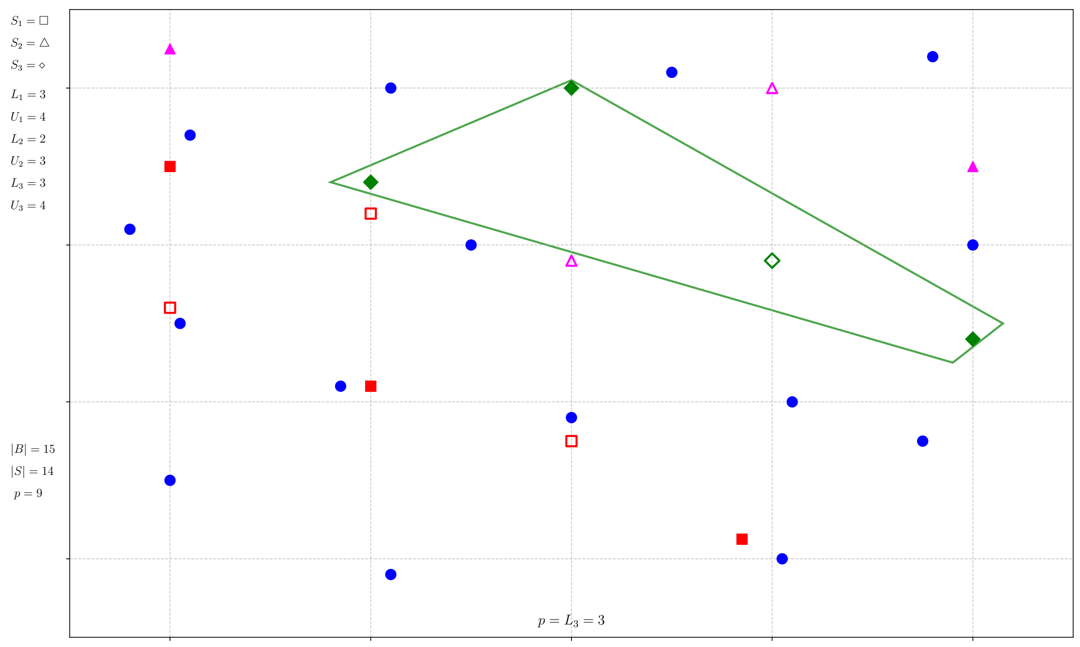


Figure A.4: Illustrative instance of the problem for p-dispersion for $k = 3$

After it is done with the subproblems, it combines the solutions into a larger set, and checks if the cardinality of the solution set is equal to the original p of the problem, which in this case is not, as it is missing one center. Therefore, it finally solves another p-dispersion problem, now taking all of the centers into account, adding centers to the solution only if their location does not produce an infeasible solution when taking into account U_k, in this case, it decides to add a new center of type 1, as shown in the following figures.

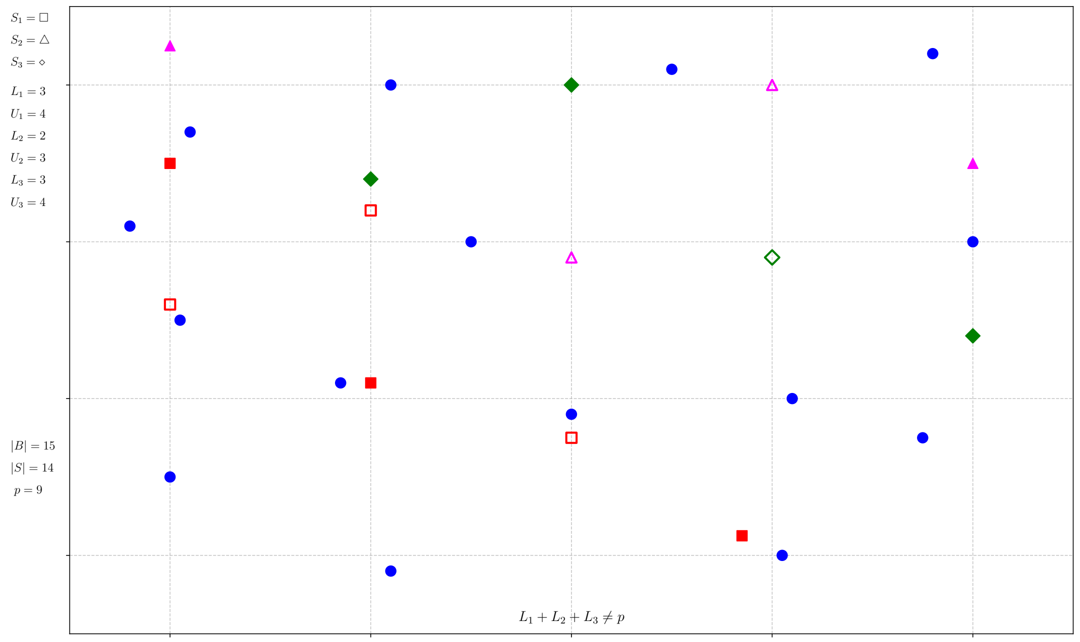


Figure A.5: Illustrative instance of the problem with the subproblems joined

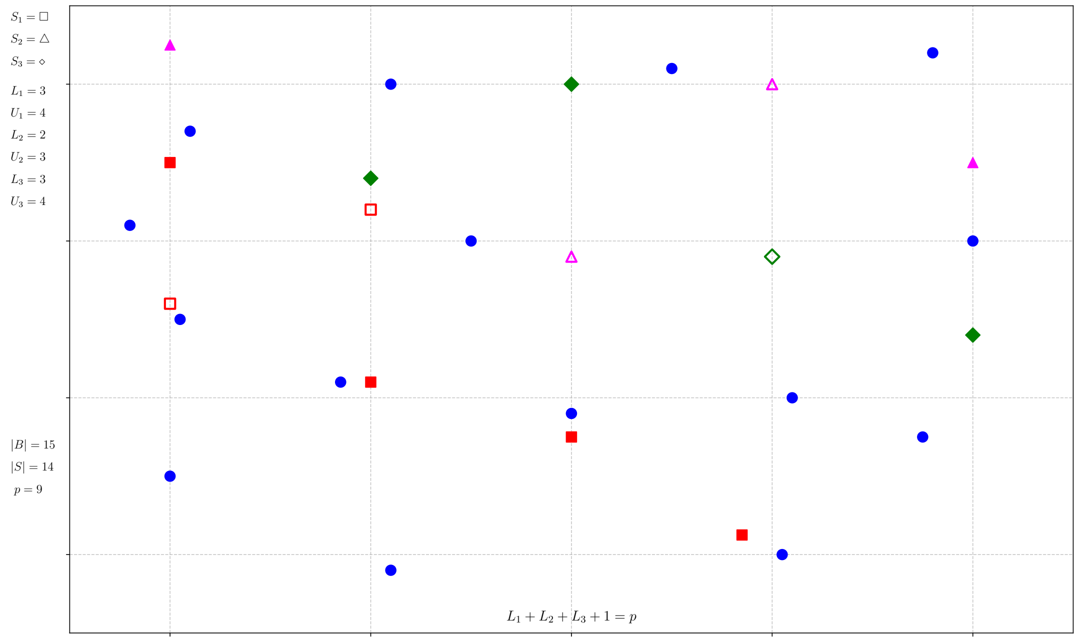


Figure A.6: Illustrative instance of the problem for p-dispersion as a whole

A.6.2 Allocation algorithm with opportunity cost

Let us assume an instance of the problem with 4 BUs and 2 centers, both opened, $Y_1 = 1, Y_2 = 1$, with the opportunity cost queue set as $\{1, 4, 2, 3\}$

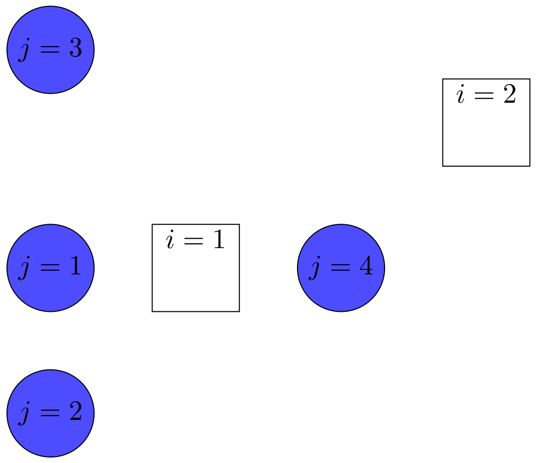


Figure A.7: Initial configuration

We dequeue the BU of $j = 1$ as it is the first element of the queue, and we assign it to its nearest center. We perform the calculation of the constraints of activity measures for $i = 1, j = 1$ and update the state of those constraints to the center, with $|M| = 3$. As a reminder, the description of the mathematical model is available in Section 3.3.

Let us assume the following values for $j = 1$: $v_1^1 = 10, v_1^2 = 30, v_1^3 = 23$ and the values of the constraints of the lower bounds for $i = 1$: $\mu_1^1 Y_1(1 - t^1) = 30, \mu_1^2 Y_1(1 - t^2) = 56, \mu_1^3 Y_1(1 - t^3) = 58$. We check if any constraint has reached the minimum value required, in this case, the center has not reached those values as seen in: $10 \leq 30$ for $m = 1$, $30 \leq 56$ for $m = 2$, $23 \leq 58$ for $m = 3$.

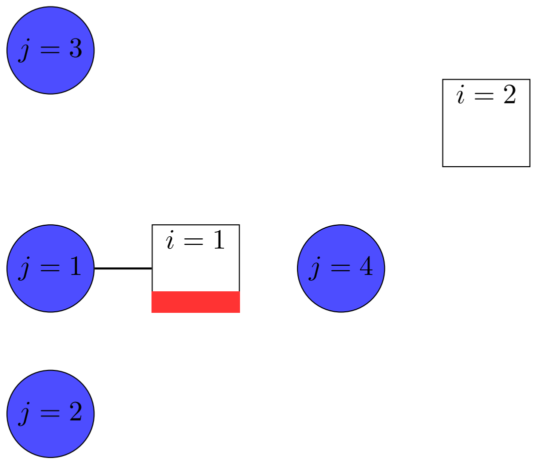


Figure A.8: First step with capacity constraints updated for $i = 1, j = 1$

We then dequeue the next BU, which is $j = 4$. The values of the activity measures that the BU provide are as follows: $j = 4$: $v_4^1 = 21, v_4^2 = 28, v_4^3 = 35$ and we update the values of $i = 1$: $10 + 21 > 30$ for $m = 1$, $30 + 28 > 56$ for $m = 2$, $23 + 35 > 58$ for $m = 3$. In this case, all activity measures have reached the minimum value required in the constraint, therefore we update the capacities of $i = 1$ and mark it as unavailable for future allocations.

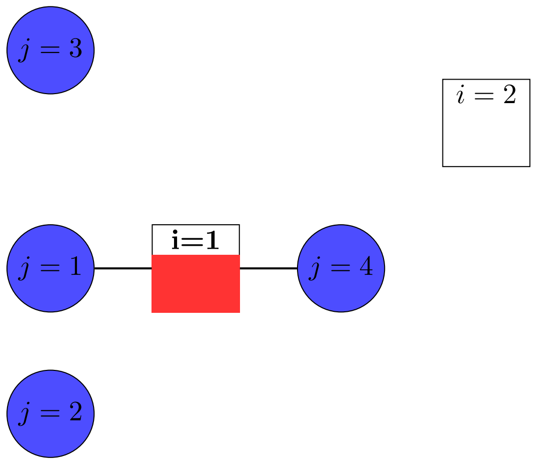


Figure A.9: Second step showing allocation from $j = 4$ to $i = 1$, unavailable.

We dequeue the next element, $j = 2$. In this case, its nearest center is $i = 1$, but it is unavailable to be allocated to, as it now complies with the constraints of the lower

bounds for the activity measures target, this forces other centers to be used, guiding the solution towards feasibility. As such, we allocate $j = 2$ to $i = 2$ and update the values for that center, with the values of the BU being: $v_2^1 = 15, v_2^2 = 37, v_2^3 = 25$, the constraints of the center being $\mu_1^1 Y_2(1-t^1) = 40, \mu_1^2 Y_2(1-t^2) = 50, \mu_1^3 Y_2(1-t^3) = 69$, with the state of the constraints being : $15 \leq 40$ for $m = 1$, $37 \leq 50$ for $m = 2$, $25 \leq 69$ for $m = 3$.

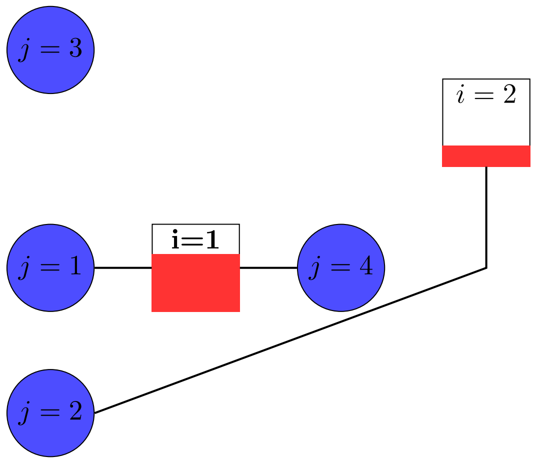


FIGURE A.10: Third step showing allocation from $j = 2$ to $i = 2$

Finally, we allocate $j = 3$ to $i = 2$, as it is the last element in the queue. We update the center capacities with the values from the BU $j = 3$: $v_3^1 = 26, v_3^2 = 30, v_3^3 = 48$, with the state of the constraints being : $15 + 26 > 40$ for $m = 1$, $37 + 30 > 50$ for $m = 2$, $25 + 48 > 69$ for $m = 3$. This makes $i = 2$ unavailable for future centers.

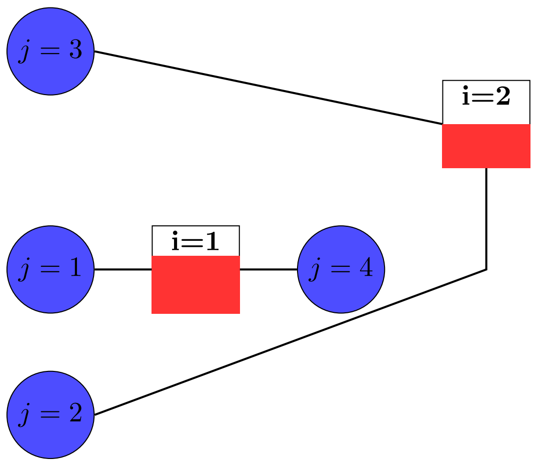


FIGURE A.11: Fourth step showing allocation from $j = 3$ to $i = 2$

The final assumption is that we introduce a fifth BU, $j = 5$, to illustrate what would happen in the case that all centers are unavailable due to their activity measures constraints.

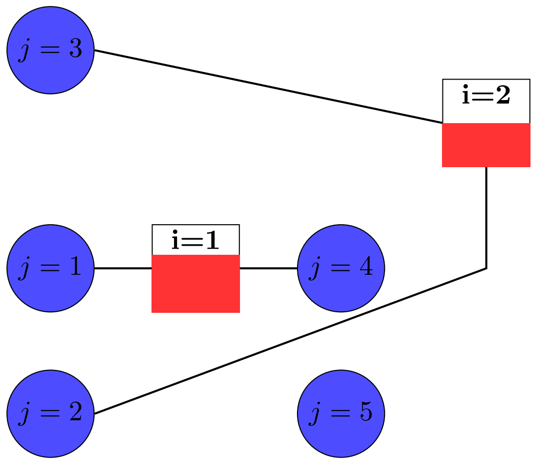


FIGURE A.12: Fifth step introducing $j = 5$

To handle unassigned BUs, we then switch to the risk threshold as the criterion of allocation, assigning the 5th BU to the center with the highest residual capacity in the risk threshold. By calculating the current used capacity in the risk threshold for each center, summing the values of the risks provided by each BU, we obtain the following results: we sum the risk values for $j = 1, j = 4$ as they are assigned

to $i = 1$, with a risk threshold of $\beta_1 = 120$, so the residual capacity for $i = 1$ is $r_1 = 40, r_4 = 15$, $120-(40+15) = 65$, for $i = 2$ with a risk threshold of $\beta_2 = 150$ we find that the assignments are $j = 2$ and $j = 3$, $r_2 = 30, r_3 = 60$, then the capacity is $150-(30+60) = 60$. In this case, $i = 1$ has the highest residual capacity in the risk threshold, so we assign $j = 5$ to it, and perform one last update on the state of the constraints, as shown in the increase of the red square representing the capacities used.

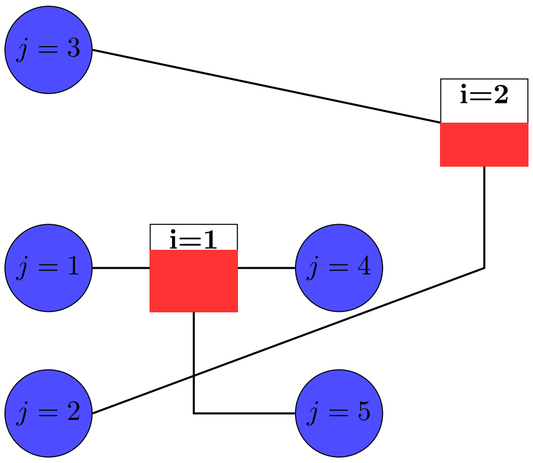


Figure A.13: Sixth step showing allocation from $j = 5$ to $i = 1$

Bibliography

G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483–485, New York, USA, 1967. Association for Computing Machinery.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2014.

G. Bruton, S. Khavul, D. Siegel, and M. Wright. New financial alternatives in seeding entrepreneurship: Microfinance, crowdfunding, and peer–to–peer innovations. *Entrepreneurship Theory and Practice*, 39(1):9–26, 2015.

J. Cano-Belmán, R. Z. Ríos-Mercado, and M. A. Salazar-Aguilar. Commercial territory design for a distribution firm with new constructive and destructive heuristics. *International Journal of Computational Intelligence Systems*, 5(1):126–147, 2012.

I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

E. Erkut and S. Neuman. Comparison of four models for dispersing facilities. *INFOR: Information Systems and Operational Research*, 29(2):68–86, 1991.

E. Erkut, Y. Ülküsal, and O. Yeniçerioğlu. A comparison of p-dispersion heuristics. *Computers & Operations Research*, 21(10):1103–1113, 1994.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

E. Fernández, J. Kalcsics, S. Nickel, and R. Z. Ríos-Mercado. A novel maximum dispersion territory design model arising in the implementation of the WEEE-directive. *Journal of the Operational Research Society*, 61(3):503–514, 2010.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL : `https://www.gurobi.com`.

P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

S. W. Hess, J. B. Weaver, H. J. Siegfeldt, J. N. Whelan, and P. A. Zitlau. Nonpartisan political redistricting by computer. *Operations Research*, 13(6):998–1006, 1965.

J. Kalcsics. Towards a unified territorial design approach - applications, algorithms and gis integration. *TOP*, 13:1–56, 2005.

J. Kalcsics and R. Z. Ríos-Mercado. Districting problems. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, chapter 25, pages 703–741. Springer, Cham, Switzerland, 2nd edition, 2019.

J. Kallrath. *Modeling Languages in Mathematical Optimization*. Springer, New York, USA, 2004.

R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

S. Khandker. Microfinance and poverty: Evidence using panel data from Bangladesh. *World Bank Economic Review*, 19:263–286, 02 2005.

S. Koranne. Hierarchical data format 5: Hdf5. In *Handbook of Open Source Tools*, pages 191–200. Springer, New York, USA, 2011.

G. Laporte, S. Nickel, and F. Saldanha Da Gama, editors. *Location Science*. Springer, Cham, Switzerland, 2019.

J. F. López, T. Ekin, F. Mendez Mediavilla, and J. A. Jimenez. Hybrid heuristic for dynamic location-allocation on micro-credit territory design. *Computacion y Sistemas*, 19(4):783–804, 2015.

J. F. López, T. Ekin, F. Mendez Mediavilla, and J. A. Jimenez. Risk-balanced territory design optimization for a micro finance institution. *Journal of Industrial & Management Optimization*, 16(2):741–758, 2020.

F. A. Medrano. The complete vertex p-center problem. *EURO Journal on Computational Optimization*, 8(3-4):327–343, 2020.

M. S. R. Monteiro. Bank-branch location and sizing under economies of scale. Master's thesis, Universidade do Porto, Portugal, 2005.

M. S. R. Monteiro and D. B. M. M. Fontes. Locating and sizing bank-branches by opening, closing or maintaining facilities. In H.-D. Haasis, H. Kopfer, and J. Schönberger, editors, *Operations Research Proceedings 2005*, pages 303–308, Berlin, Germany, 2006. Springer.

D. R. Quevedo-Orozco and R. Z. Ríos-Mercado. Improving the quality of heuristic solutions for the capacitated vertex p-center problem through iterated greedy local search and variable neighborhood descent. *Computers & Operations Research*, 62: 133–144, 2015.

S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

R. Z. Ríos-Mercado and J. F. Bard. An exact algorithm for designing optimal districts in the collection of waste electric and electronic equipment through an improved reformulation. *European Journal of Operational Research*, 276(1):259–271, 2019.

R. Z. Ríos-Mercado, editor. *Optimal Districting and Territory Design*, volume 284 of *International Series in Operations Research and Management Science*. Springer, Cham, Switzerland, 2020.

R. Z. Ríos-Mercado and H. J. Escalante. GRASP with path relinking for commercial districting. *Expert Systems with Applications*, 44:102–113, 2016.

R. Z. Ríos-Mercado and E. Fernández. A reactive GRASP for a commercial territory design problem with multiple balancing requirements. *Computers & Operations Research*, 36(3):755–776, 2009.

R. Z. Ríos-Mercado and J. F. López-Pérez. Commercial territory design planning with realignment and disjoint assignment requirements. *Omega*, 41(3):525–535, 2013.

R. Z. Ríos-Mercado, A. M. Álvarez Socarrás, A. Castrillón, and M. C. López-Locés. A location-allocation-improvement heuristic for districting with multiple-activity balancing constraints and p-median-based dispersion minimization. *Computers & Operations Research*, 126:105106, 2021.

M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and M. Cabrera-Ríos. New models for commercial territory design. *Networks and Spatial Economics*, 11(3):487–507, 2011.

M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and J. L. González-Velarde. GRASP strategies for a bi-objective commercial territory design problem. *Journal of Heuristics*, 19(2):179–200, 2013.

M. G. Sandoval, J. A. Díaz, and R. Z. Ríos-Mercado. An improved exact algorithm for a territory design problem with p-center-based dispersion minimization. *Expert Systems with Applications*, 146:113150, 2020.

J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*. Wiley, New York, USA, 2003.

E. D. Taillard. *Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem.* Graduate Texts in Operations Research. Springer International Publishing, Cham, Switzerland, 2023.

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Acta de Titulación

Institución 190001
Programa 501363

642373/1847972/63753

52094
Acta Núm. ....

En la Ciudad de Monterrey, capital del Estado de Nuevo León, al día 25 del mes de Junio del año 2024, siendo las 12:00 horas, reunidos en las instalaciones de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA de la Universidad Autónoma de Nuevo León, los(as) profesores(as) DR. ROGER ZIRAHUEN RÍOS MERCADO, DRA. MARÍA ANGÉLICA SALAZAR AGUILAR, DR. VINCENT ANDRÉ LIONEL BOYER , quienes fueron designados por la dirección de la escuela o facultad para integrar el Comité de Titulación de EDUARDO SALAZAR TREVIÑO, quien cursó y aprobó todas y cada una de las unidades de aprendizaje del Programa Educativo de LICENCIATURA COMO INGENIERO EN TECNOLOGÍA DE SOFTWARE, tal como lo dispone la Ley Orgánica de la Universidad Autónoma de Nuevo León publicada en el Periódico Oficial el siete de junio de mil novecientos setenta y uno, y en su Título Quinto: De la titulación, Capítulo I, De la obtención del título, del Reglamento para la Admisión, Permanencia y Egreso de los Alumnos de la Universidad Autónoma de Nuevo León, aprobado el 8 de agosto de 2019 y en el Reglamento Interno de la Escuela o Facultad.

Se procedió a tomar la Protesta de Ley por el Presidente del Comité de Titulación y en cumplimiento de lo dispuesto por los preceptos legales y reglamentarios, firman la presente acta los profesores(as), ante la presencia del Secretario del Comité que da fe.

PRESIDENTE	SECRETARIO
[illegible]	[illegible]
DR. ROGER ZIRAHUEN RÍOS MERCADO	DRA. MARÍA ANGÉLICA SALAZAR AGUILAR

VOCAL

[illegible]

DR. VINCENT ANDRÉ LIONEL BOYER

El suscrito, Director de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA, CERTIFICA que las firmas que aparecen en la presente acta son auténticas y las mismas que utilizan los C.C. profesores mencionados en ella.

U.A.N.L.
DIRECCIÓN

[illegible]

DR. ARNULFO TREVIÑO CUBERO

El C. Secretario General de la Universidad Autónoma de Nuevo León, CERTIFICA que la firma del C. Director de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA que aparece en la presente acta, es auténtica.

[illegible]

DR. JUAN PAURA GARCÍA

SECRETARÍA GENERAL
DEPARTAMENTO ESCOLAR Y DE ARCHIVO

jmor/esb **El recibir de conformidad este documento académico, compromete al interesado a dar el uso legal y correcto del mismo. Quien altere cualquiera de las partes que lo conforman, invalida inmediatamente su autenticidad, haciéndose acreedor a la aplicación de las sanciones previstas en las leyes y reglamentos de la UANL; independientemente de los efectos legales que procedan. IA 24/06/24**

UANL-T
013744

ORIGINAL

LPT-23

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

ingeniería en tecnología de software

.

Junio 2024

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

INGENIERÍA EN TECNOLOGÍA DE SOFTWARE

.

Junio 2024

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Los miembros del Comité de Tesis recomendamos que la Tesis "A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions", realizada por el alumno Eduardo Salazar Treviño, con número de matrícula 1847972, sea aceptada para su defensa como requisito para obtener el grado de Ingeniería en Tecnología de Software.

El Comité de Tesis

Dr. Roger Z. Ríos Mercado
Director

Dra. María Angélica Salazar Aguilar
Revisor

Dr. Vincent André Lionel Boyer
Revisor

Vo. Bo.

Dr. Fernando Banda Muñoz
Subdirector Académico

San Nicolás de los Garza, Nuevo León, Junio 2024

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Los miembros del Comité de Tesis recomendamos que la Tesis "A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions", realizada por el alumno Eduardo Salazar Treviño, con número de matrícula 1847972, sea aceptada para su defensa como requisito para obtener el grado de Ingeniería en Tecnología de Software.

El Comité de Tesis

[illegible]

Dr. Roger Z. Ríos Mercado
Director

[illegible]

Dra. María Angélica Salazar Aguilar
Revisor

[illegible]

Dr. Vincent André Lionel Boyer
Revisor

Vo. Bo.

[illegible]

Dr. Fernando Banda Muñoz
Subdirector Académico

San Nicolás de los Garza, Nuevo León, Junio 2024

A los gigantes

en cuyos hombros estoy parado.

Contents

List of Figures

List of Tables

Agradecimientos

Agradezco a mis señores padres, Eduardo Salazar Naranjo y Laura Marcela Treviño Lara, por haberme apoyado en cumplir todos mis sueños y metas. Ahora que culmino mi primera etapa de estudios profesionales quiero agradecerles todo lo que han hecho por mí hasta llegar a este punto, se los debo todo a ustedes, las palabras no son suficientes para darles las gracias por su sacrificio y esfuerzo durante toda mi vida, que me han permitido soñar y cumplir esos sueños. Agradezco también a mis hermanas, Carla Elizabeth y María José por su apoyo moral y amistad fraternal, a lo largo de toda mi vida y en especial en mi carrera.

Quiero agradecer a mi principal asesor, director de tesis y mentor, el Doctor Roger Z. Ríos Mercado por toda su guía brindada en este proceso. Ha sido un camino muy largo el que he recorrido siendo impulsado y guiado por el. No existen palabras para expresar la gratitud que siento con y hacia el por su infinita paciencia, sabiduría, comentarios y retroalimentación de estos dos años. Nada de esto sería posible sin él. Adicionalmente quiero agradecer profundamente también a la Doctora Diana Lucía Huerta Muñoz, también mi coasesora por todos sus comentarios y ánimos en el proceso, si algún día sentía que iban mal las cosas, la doctora. siempre me podía animar con su optimismo, sus comentarios y atención al detalle han sido invaluables en este proceso.

Gracias a mis revisores, la doctora María Angélica Salazar Aguilar y al doctor Vincent André Lionel Boyer por su tiempo y esfuerzo en la revisión y mejora de este trabajo.

Agradezco a la Facultad de Ingeniería Mecánica y Eléctrica el haberme permitido ser alumno de la mejor institución ingenieril de México, a la Universidad Autónoma de Nuevo León por acogerme como alumno desde el bachillerato y brindarme las herramientas y oportunidades que hoy estoy aprovechando. También agradezco al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) por su apoyo económico con la beca de Ayudante de Investigador del SNI nivel III y a la clase trabajadora de México, con cuyos impuestos recaudados se financían estos programas.

Asimismo a pesar de no formar parte oficialmente del Programa del Posgrado en Ingeniería de Sistemas (PISIS) en la FIME, quiero agradecer también a sus maestros y alumnos por también haber colaborado en este proceso, la doctora Elisa, la doctora Sara Elena, el maestro Said y el doctor Sergio.

Por último, gracias a todos los amigos que he hecho en el camino: Neto, Pablo, Chava, Saúl, Emmanuel, Isaac, Uriel, Betty, Tali y Rodolfo.

Resumen

Eduardo Salazar Treviño.

Candidato para obtener el grado de Ingeniería en Tecnología de Software.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Número de páginas: 135.

Objetivos y método de estudio: Analizar, diseñar e implementar algoritmos heurísticos eficientes que proporcionen soluciones de buena calidad para un problema de optimización combinatoria de diseño territorial de una institución microfinanciera, así como comparar los resultados y tiempos de ejecución de los algoritmos propuestos con respecto a los resultados obtenidos por software de optimización exacta.

El problema de diseño territorial se refiere a la partición de un conjunto B de unidades geográficas básicas y un conjunto S de posibles centros territoriales en p territorios, respetando restricciones espaciales y de planificación.

Contribuciones y conclusiones: En esta tesis se ha desarrollado e implementado una metaheurística basada en un Procedimiento de Búsqueda Adaptativa Ávida Aleatorizada (*GRASP* en inglés) para abordar el problema de estudio, demostrando eficacia al obtener soluciones comparables a las mejores soluciones encontradas por software de optimización exacta y con mejor tiempo de cómputo.

Firma del director: ______________________
Dr. Roger Z. Ríos Mercado

Abstract

Eduardo Salazar Treviño.

As a candidate to obtain a degree in Software Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of the study: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Number of pages: 135.

Objectives and Study Methods: The purpose of this thesis is to analyze, design, and implement efficient heuristic algorithms that provide good quality solutions for a combinatorial optimization problem arising in the territorial design for a microfinance institution.

The territorial design problem refers to the partitioning of a set B of basic geographical units and a set S of possible territorial centers into p territories. This is done while meeting spatial and planning constraints.

Contributions and Conclusions: A Greedy Randomized Adaptive Search Procedure (GRASP) has been developed and implemented to address the problem under study, demonstrating efficiency in obtaining solutions comparable to the best solutions found by exact optimization algorithms, with shorter running times.

Director signature: ______________________________
Dr. Roger Z. Ríos Mercado

Resumen

Eduardo Salazar Treviño.

Candidato para obtener el grado de Ingeniería en Tecnología de Software.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Número de páginas: 114.

Objetivos y método de estudio: Analizar, diseñar, e implementar algoritmos heurísticos eficientes que proporcionen soluciones de buena calidad para un problema de optimización combinatoria de diseño territorial de una institución microfinanciera, así como comparar los resultados y tiempos de ejecución de los algoritmos propuestos con respecto a los resultados obtenidos por software de optimización exacta.

El problema de diseño territorial se refiere a la partición de un conjunto B de unidades geográficas básicas y un conjunto S de posibles centros territoriales en p territorios, respetando restricciones espaciales y de planificación.

Contribuciones y conclusiones: En esta tesis se ha desarrollado e implementado una metaheurística basada en un Procedimiento de Búsqueda Adaptativa Ávida Aleatorizada (*GRASP* en inglés) para abordar el problema de estudio, demostrando eficacia al obtener soluciones comparables a las mejores soluciones encontradas por software de optimización exacta y con mejor tiempo de cómputo.

Firma del director: [illegible]
Dr. Roger Z. Ríos Mercado

Abstract

Eduardo Salazar Treviño.

As a candidate to obtain a degree in Software Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of the study: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Number of pages: 114.

Objectives and Study Methods: The purpose of this thesis is to analyze, design, and implement efficient heuristic algorithms that provide good quality solutions for a combinatorial optimization problem arising in the territorial design for a microfinance institution, as well as comparing the results and running times of the proposed algorithms with respect to the results obtained by exact optimization software.

The territorial design problem refers to the partitioning of a set B of basic geographical units and a set S of possible territorial centers into p territories. This is done while meeting spatial and planning constraints.

Contributions and Conclusions: A Greedy Randomized Greedy Adaptive Search Procedure (*GRASP*) has been developed and implemented to address the problem under study, demonstrating efficiency in obtaining solutions comparable to the best solutions found by exact optimization software, with shorter run-times.

Director signature: [illegible]
Dr. Roger Z. Ríos Mercado

Chapter 1

Introduction

A classic Territorial Design Problem (TDP), also known as a Districting Problem, organizes and partitions a given set B of small geographical units or *Basic Units* (BUs), and a set S of territory centers into p larger geographical clusters called territories or districts according to spatial constraints such as compactness and contiguity, and planning constraints in order to balance the magnitude of the performance of several activities across all territories, such as their economic performance (sales, total workload, number of customers served) or specific physical needs of each center (Ríos-Mercado and López-Pérez, 2013).

A district is called *contiguous* if it is possible to travel between each pair of basic units of the district without leaving the district. *Compactness* refers to round-shaped, undistorted territories without holes. The reason for preferring compact districts is the need of maintaining contiguity, as it minimizes daily travel distances within the territories. In terms of compactness, a dispersion measure based on the objective function of several classic location problems, such as the p-center problem or the p-median problem, is usually considered (Laporte, Nickel, and Gama, 2019). Since p is not equal to $|S|$, the first decision to be taken is which of all the S possible centers must be used to serve the BUs. Once the p centers are known, the individual allocation of BUs to their center can be decided.

1.1 Motivation

The purpose of this study is to provide a generalization for microfinancial institutions that wish to design territories representing where to open a service branch and which clients will be served by that branch. Risk balancing among the branches is extremely important for this type of business, as a failure in a single branch node can compromise the entire network.

The model described in this thesis is motivated by a previous model described in López, Ekin, Mediavilla, and Jimenez (2020). In the present model, risk balancing is modeled as a constraint, whereas in the original model it is a part of the objective function. Moreover, the model from López et al. (2020) was applied to an existing territorial design, so the objective function also included a term related to retaining as much as possible the same territorial features of the original design. Additionally, in our model we assume we do not have connectivity constraints. Finally, the previous model is solved using its mathematical formulation as a Mixed Integer Linear Programming problem.

The proposed model can be viewed as a p-median problem with multiple capacity constraints. Given that even the uncapacitated p-median problem is $\mathcal{NP}$-hard (Ríos-Mercado and Escalante, 2016), our TDP is also $\mathcal{NP}$-hard. The $\mathcal{NP}$-hard nature of the problem is what motivates us to pursue the use of heuristic approaches for solving it in a reasonable time.

1.2 Objectives

- Studying and analyzing a generalization of a territorial design problem with activity measures, risk, and center types balancing constraints.
- Designing and implementing efficient and robust heuristic and metaheuristic

algorithms for the problem.

- Demonstrating the effectiveness of solving such a problem using the proposed heuristic framework.

- Comparing the performance of the proposed robust metaheuristic with respect to a state-of-the-art exact solver.

1.3 Contribution

In this work, a Greedy Randomized Adaptive Search Procedure metaheuristic is proposed to efficiently solve a territorial design problem of a microfinancial institution.

The key components of this metaheuristic are: a robust constructive heuristic which guarantees near-feasible solutions, repaired later by a local search phase. Once a solution is feasible, the local search changes its priority towards improving the objective function using several local neighborhood structures considering a cyclical neighborhood exploration strategy. The proposed metaheuristic framework which involves parallel computing has been assessed and evaluated over a wide range of test instances.

By comparing the results of the proposed metaheuristic versus the solutions obtained from the mathematical model solved by a general-purpose optimization solver, we observe that the heuristic outperforms off-the-shelf optimizers in terms of total computing time and solution quality is within an acceptable relative optimality gap to the best bound found by the solver.

This thesis is organized as follows: In Chapter 2, we provide the literature review related to the problem. In Chapter 3, the definition of the problem and the mathematical formulation are presented. In Chapter 4, the proposed algorithms are described. Chapter 5 presents the computational experience where the proposed

metaheuristic is fully assessed. Finally, in Chapter 6, we provide the conclusions for this research work.

Chapter 2

Literature Review

In this chapter, specifically in Section 2.1, we analyze the literature for the Territorial Design Problem (TDP) and an overview on microfinancial institutions and their importance. Sections 2.2 and 2.3 describe the problem and provide the representation of feasible solutions. In Section 2.4, we will model the problem based on the previous description.

This TDP identifies potential branch offices (territory centers) and allocates clients to their respective office, balancing several constraints while minimizing the distance of the clients and their assigned institution branch.

The territory centers must be distributed across different hosts' business lines. Due to the *micro* nature of the institution, there are not enough resources to build new physical offices to give service, instead, those offices must be opened in existing facilities of different businesses and services, such as restaurants, gas stations, pharmacies, etcetera, to serve clients alongside the main activity performed by the establishment. Therefore, all centers must be distributed evenly across the different hosts' business lines.

The aim is to determine what centers to open from a set of possible locations, such that the sum of distances from the basic units to their assigned institution branch is minimized, subject to balance criteria: the required workload for each

BU, total quantity of money involved in the loans for each BU, and total utilities generated for each BU. These criteria must be balanced around a target value, and since the exact value will not be met perfectly, a tolerance parameter is introduced to allow some leeway. Finally, the sum of the risk involved in each loan for every center must not exceed a given threshold.

2.1 Districting

Districting problems (DP) or territorial design problems arise in many different contexts; from classical areas such as political districting, sales alignment, or public service, to more recent applications in police districting, waste management, commercial or healthcare (Ríos-Mercado, 2020).

The challenge of establishing political boundaries can be perceived as partitioning an administrative region, like a city or state, into subdivisions for electing political representatives, often known as *political districting.* This issue holds significant importance in democratic systems where each district selects representatives for a legislative body. To avoid politicization of the redistricting process, numerous states have established legislative and practical standards. Several criteria are used to make the districting decisions: demographic criteria such as population, minority representation and voter equality; geographic criteria that model the previously discussed compactness and contiguity features and, finally, political data.

The crucial but arduous work of developing sales and services territories is shared by all businesses that employ a sales team and require the market area to be divided into zones of accountability. Similarly, the issue of defining service territories for customer or technical facility support is closely related. In these cases, similar standards are typically used to create regions for personnel of service. There are multiple reasons for adjusting current territories, or for planning new territories. First, any increase or decrease in the number of sales or service personnel undoubtedly

requires adjustments to the territories. Additional justifications include enhancing the current workforce's reach or to distribute workload uniformly among them. Furthermore, shifts in customer demographics or the roll-out of new products require a restructuring of boundaries.

The sales territorial design problems are closely related to the problem addressed in this work. Several criteria are overlapped between both of them, for instance, the number of territories is predetermined and the presence of basic units, that represent individual customers assigned to larger geographical units with exclusive assignments are considered. Sales districting problems also have activity-related criteria that represent the performance of the products or service that an organization may offer. Dealing with sales and service territory districting issues, there is often a pursuit for districts that balance one or more characteristics (referred to as *activity measures*). This standard shows the relationship between territories to ensure equitable treatment of all sales personnel (Kalcsics, 2005).

The creation of service districts emerges in multiple scenarios. One of them is focused on social infrastructures: hospitals or public utilities. There are cases where districts are required to assign each resident to a specific facility for service provision, such as routine health check-ups, or to delimit areas of responsibility for home-care visits by healthcare staff such as nurses or physiotherapists. The objective is to identify joint districts that are easily reachable via public transport and ensure a balanced workload based on service and travel time or aim for optimal capacity usage of the social facility.

Despite all the different criteria of all the possible applications of districting, there is no definitive mathematical model to represent each problem. Depending on each context, the formulation of the problem and the solution approach are different. The compactness criterion is also something that depends on the specific problem as there is no concrete definition of compactness since it depends on the representation of the basic units. In political districting, compactness is preferred to be represented

as specific territorial shapes, while in sales and services districting, distance-based compactness is preferred.

For comprehensive surveys and discussion on districting models, algorithms, and applications, the reader is referred to the works by Kalcsics and Ríos-Mercado (2019) and Ríos-Mercado (2020).

The studied TDP considers a distance-based dispersion measure, which can be represented in several manners, resulting in several models and solutions. The commonly used dispersion measures are those used for the p-Median Problem (pMP) and the p-Center Problem (pCP). In the pMP, the objective is to place p facilities (or medians) in such a way that the total weighted distance (or cost) between demand points and their assigned facilities is minimized (Laporte, Nickel, and Gama, 2019). This is particularly suitable for situations where the total travel (or service) cost needs to be minimized, for example: to determine the optimal locations for warehouses to minimize the total transportation cost. In contrast, the pCP aims to locate p facilities (or centers) such that the maximum distance from a demand point to the nearest facility is minimized (Medrano, 2020). This essentially minimizes the worst-case scenario in terms of the distance traveled by a customer to reach a facility. This might be used, for example, to locate emergency facilities such as fire stations or hospitals, where the aim is to minimize the longest distance that must be traveled in an emergency. In the context of districting, Salazar-Aguilar et al. (2011) present a computational study of both pMP and pCP objectives used within a territory design framework. For this thesis, the dispersion is measured using a function from the p-Median problem.

2.2 Microfinancial institutions

Microfinancial institutions are important to reduce income inequality and poverty (Khandker, 2005). They offer credit and other financial products to sectors of the

population that are normally rejected by commercial banks as they represent high-risk with low profit options. They have risen as an alternative to regular banking, especially in countries with developing economies (Bruton, Khavul, Siegel, and Wright, 2015). Due to their fragile nature, the risk associated with the loans granted must be balanced across the branch offices.

Compared to traditional banking systems, the MFIs do not operate nor own branch offices, instead, only overseeing credit operations. These branch offices or subsidiaries operate as part of local businesses (gas stations or grocery stores) that allow MFIs to reduce operating and construction costs. MFIs face risks that normal banks do not. Traditional banking systems discourage the practice of lending money to people who do not have guarantees to secure the loans and are unable to pay them back, excluding vulnerable sectors of the population.

A relationship of mutual rewards with the branches is established so that the network remains sustainable in the long term. Balancing risks between different branches is important, since there is no universal definition of risk, measures can be based on statistical deviations or problem-specific functions.

In this thesis, a function based on the problem's variance is used, which will be referred to as the variance of excessive gain. For each BU, a gain variance corresponds to a higher risk in which small establishments are not exposed to high variance. Furthermore, each open branch office and their corresponding served clients, forms a territory or district, with the office being the territory center and the clients the basic units.

Also, each type of branch must manage different levels of risk in order to control and balance the tolerance of each territorial center. For this, the gain variance for each BU is first estimated using historical customer data. Then, the total gain variance within the centers is used as an internal risk measure to obtain information about the customer retention rate. This rate estimates how probable it is that a client continues to be such. Therefore, the MFI pre-determines an upper limit for

the total weighted gain variance, for each territorial center. Subsequently, customers should be assigned to territories in such a way that the total variance of excessive gain is minimized. Deviations towards lower limits are not taken into account. A more detailed description of the studied problem in this thesis is provided below.

2.3 Districting applied to MFIs

Territorial design models are rarely applied to traditional banking in the literature. The most important contribution to the literature can be found in Monteiro (2005) and Monteiro and Fontes (2006), in which traditional MILP formulations are applied to locating and maintaining bank-branches, closing or opening existing branches by posing a coverage problem. However, these contributions recognize the shortcomings of the MILP approach and introduce local search heuristics that start from a partial solution provided by an optimization solver. These contributions take into consideration several costs of operation, location, as well as sizing, thus they are not exactly districting formulations.

Districting, specifically applied to MFIs, first appeared in López, Ekin, Mediavilla, and Jimenez (2015) where a two phased mixed integer program, hybridized with several heuristics, is used to solve a real-life instance from a MFI located in Monterrey, México. This model presents the risk balancing as a parameter in the objective function as well as introducing a similarity parameter, which motivates the new design to retain the features of a previous design. Additionally, another constraint limits the maximum amount of distance traveled by the customers. Some BUs are constrained to be allocated to different territories, introducing disjoint assignment constraints. Finally, contiguity constraints, which evaluate that every district induces a connected graph between all the BUs and the corresponding territory center, are included. This type of constraint in specific grows exponentially, so solving medium to large instances becomes intractable in a reasonable computing time. By using heuristics in several phases, they were able to reduce their problem complexity

in order to use an exact solver.

A follow-up work was published in López, Ekin, Mediavilla, and Jimenez (2020), in which the heuristics used to set some variables for the solver were slightly improved on, using Geographic Information System (GIS) software to provide interpretability to the MFI and properly implement the solution found. As it can be seen, there have been no previous attempts to apply districting to MFIs in a broader context, not tied to a specific case study.

In this thesis, we propose a generalization of the model of López, Ekin, Mediavilla, and Jimenez (2020) so it can be applied in more cases by removing: the hard constraints on maximum distance, the disjoint assignments constraints, and the contiguity constraints, as well as simplifying the objective function by considering the risk term as a constraint and removing the similarity to a previous plan parameter. Notice that most territorial design problems have a district center c_k for each district D_k that coincides with a basic unit, i.e. $c_k \in B$: however, in this thesis, the district center is a member of another set, $c_k \in S$. By reformulating the problem and solving it with an exact solver, we can verify that the computation is not trivial and an optimal solution cannot be reached in reasonable time; thus, we encourage the use of heuristics and metaheuristics to solve the problem. This thesis is the first work, to the best of our knowledge, to propose a generalized heuristic framework for districting applied to MFIs and, subsequently, the first to propose metaheuristics.

CHAPTER 3

PROBLEM DESCRIPTION AND MATHEMATICAL MODEL

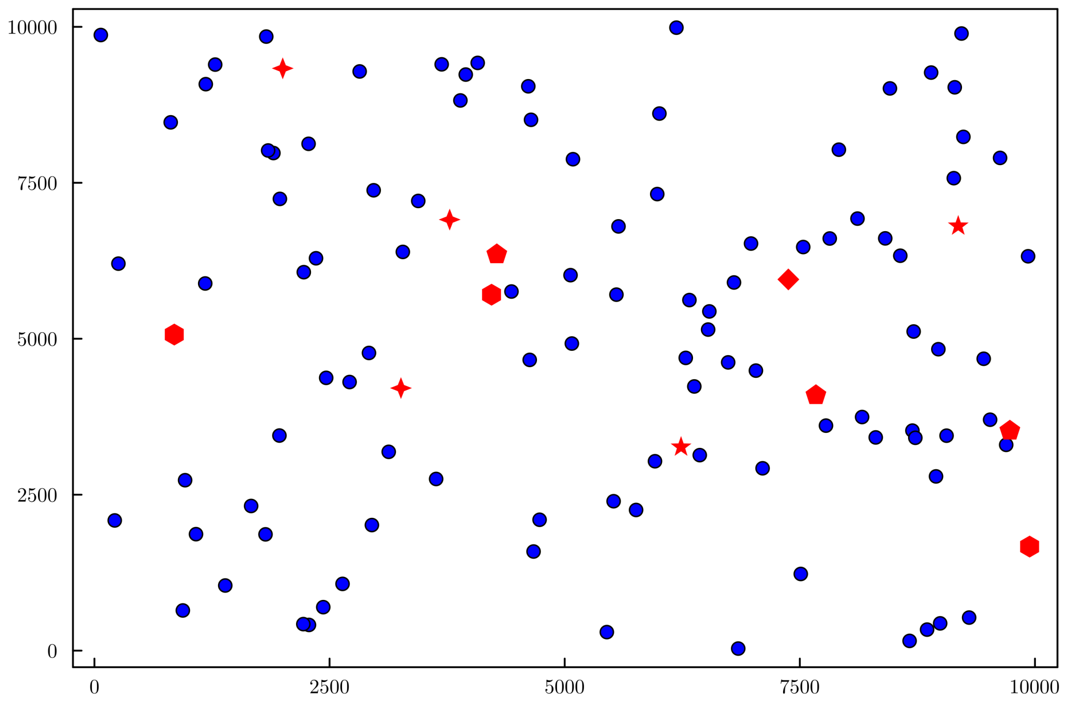
3.1 PROBLEM DESCRIPTION

Given a set S of possible territory centers, a set B of BUs, a distance matrix d and a p number of centers to be open, let the set M of activities to evaluate and their corresponding activity values v_m^j, where $m \in M$ and $j \in B$, the sum of these values for all the BUs assigned to a territory center must be in a range defined by multiplying the target measure μ_m^i for each activity $m \in M$, $i \in S$, and a t_m tolerance parameter. Additionally, given each individual risk value r_j for $j \in B$, the sum of all the risk values of the assigned BUs to a center i must not exceed a risk threshold β_i.

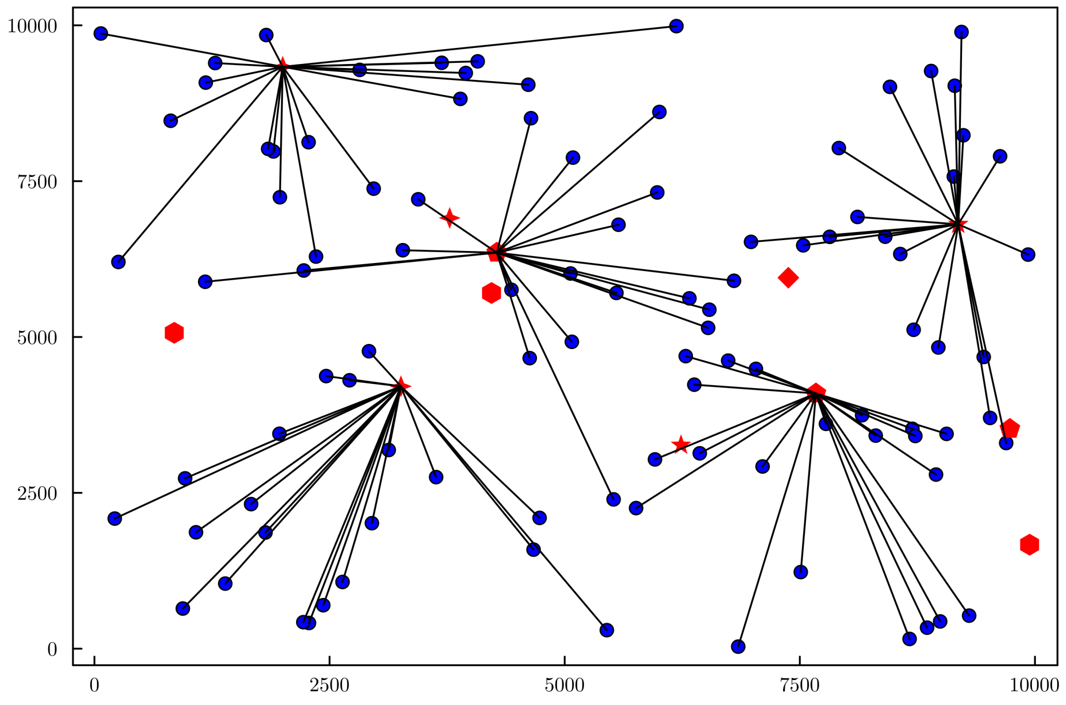
Given a set K of territory types, an associated T_{ik} value is assigned for each center $i \in S$ of type k, for $k \in K$. A lower bound L_k and an upper bound U_k, are determined for the number of centers used for each type, $\forall k \in K$. Finally, each BU $j \in B$ must be assigned only once. The territorial design problem aims to find the location of the p centers to be opened and the allocations of all BUs to their corresponding centers such that the sum of the distances in d between each BU and its assigned center is minimized, taking into account the previous constraints.

3.2 Illustrative example of feasibility

Figure 3.1a shows a basic example of an instance of the problem with BUs and territory centers, randomly located. The BUs are represented as blue circles, and the different red geometric figures are used to represent the different center types, with $K = 5$. In contrast, Figure 3.1b illustrates the optimal solution to the instance, minimizing the total distance between each BU and their assigned territory center. It is important to emphasize that there are only 5 centers available to serve BUs, as $p = 5$. In an instance of larger size, the optimal solution that minimizes the sum of distances may allocate individual BUs to centers that are not their nearest, in a counter-intuitive fashion. These allocations arise as the territories are constrained by the activity measures and the risk thresholds, as seen in Figure 3.2b.

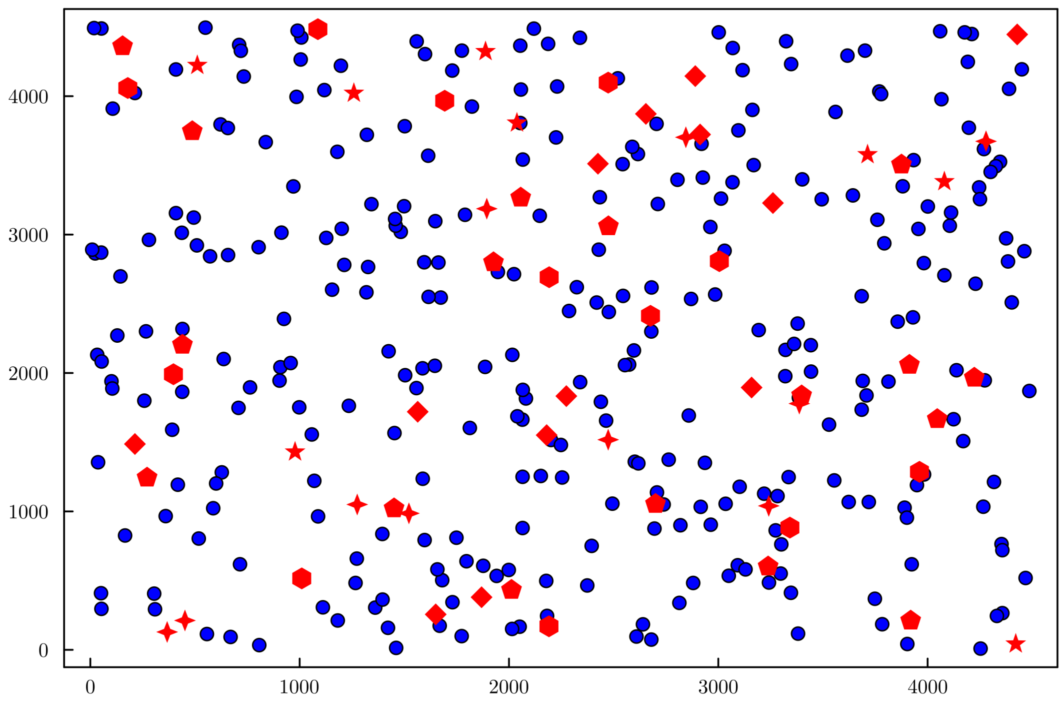


(a) Example of an instance with $|B| = 100$, $|S| = 12$, $p = 5$

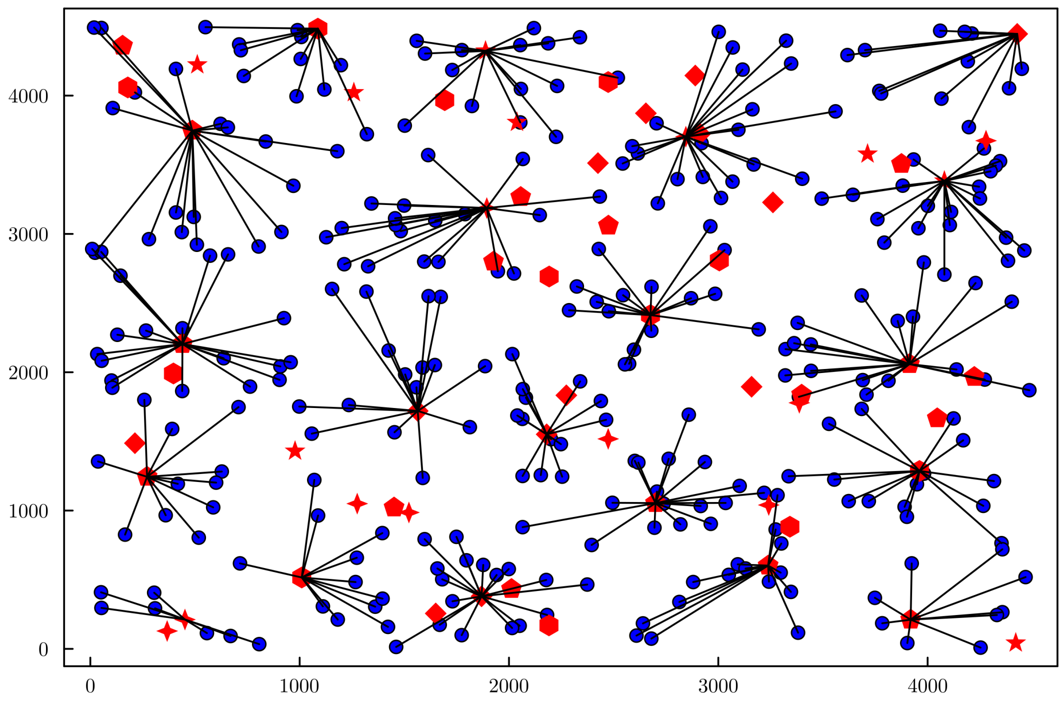


(b) The optimal solution for the instance.

Figure 3.1: Optimal solution with a weight of 208567 for a small size instance



(a) Instance with $|B| = 300$, $|S| = 60$, $p = 20$



(b) Optimal solution for the above instance.

Figure 3.2: Optimal solution for a larger instance

3.3 Mathematical model

In this section, the mathematical model used in this thesis is described, studied, and analyzed. This model is based on the formulation presented by López, Ekin, Mediavilla, and Jimenez (2020) with the following modifications. The objective function in the original model has three terms to minimize: the sum of distances, the risk of the allocations, and a similarity factor that measures how similar the new territorial design is to an existing design. As this version of the problem models the risk as a constraint and has no previous territorial design, the objective function only minimizes the distance. Additionally, the previous model has a specific connectivity constraint set that requires all territories to induce a connected graph with the BUs and their corresponding center. This set has an exponential size, thus they limit this restriction to a given set of territory centers only. On the other hand, there is also a restriction that ensures that certain pairs of BUs are allocated to different territories due to geographical or management requirements. Both of the connectivity and the exclusive assignments are not present in the current model given the additional complexity they would represent.

- Sets and parameters:

- S : Set of possible territory centers
- B : Set of possible BUs
- M: Set of activities to evaluate
- K: Set of territory center types
- d_{ij}: Distance between center i and BU j, $i \in S, j \in B$
- μ_m^i: Target of activity m measured at center i, $i \in S$, $\forall m \in M$
- v_m^j: Measure of activity m at BU j, $j \in B$, $\forall m \in M$

- t_m: Tolerance of activity m measure, $\forall m \in M$
- r_j: Risk value at BU j, $j \in B$
- β_i: Risk threshold at center i, $i \in S$
- $T_{ik} = 1$ if center i is of type k, 0 otherwise, $i \in S, k \in K$
- $L_k, U_k =$ Lower and upper bounds of number of centers of type k to be located, $k \in K$
- $p =$ Number of centers to be used

- Decision variables:

- $Y_i = 1$ if center i is open; 0 otherwise.
- $X_{ij} = 1$ if BU j is assigned to center i; 0 otherwise.

Then, the territorial design problem for the MFI is defined as:

$$\min \sum_{i \in S, j \in B} d_{ij} X_{ij} \tag{3.1}$$

Subject to

$$\sum_{i \in S} X_{ij} = 1, \qquad \forall j \in B \tag{3.2}$$

$$X_{ij} \leq Y_i, \qquad \forall i \in S, j \in B \tag{3.3}$$

$$\mu_m^i (1 - t_m) Y_i \leq \sum_{j \in B} v_m^j X_{ij} \leq \mu_m^i (1 + t_m) Y_i, \qquad \forall i \in S, \forall m \in M \tag{3.4}$$

$$L_k \leq \sum_{i \in S} T_{ik} Y_i \leq U_k, \qquad \forall k \in K \tag{3.5}$$

$$\sum_{i \in S} Y_i = p \tag{3.6}$$

$$\sum_{j \in B} r_j X_{ij} \leq \beta_i, \qquad \forall i \in S \qquad (3.7)$$

$$X_{ij} \in \{0,1\}, Y_i \in \{0,1\}, \qquad \forall i \in S, \forall j \in B \qquad (3.8)$$

The objective function (3.1) minimizes the sum of the distances between the BUs and their assigned center. Constraints (3.2) state that each BU j must be assigned to a single center i. Constraints (3.3) check that the BUs are only assigned to open centers. Constraints (3.4) establish that the activity m measured for each territory i, must be within a tolerance range defined by the t parameter. Notice that because of the discrete structure of the problem it is impossible to perfectly balance each territory. Constraints (3.5) limit the located centers' types within their corresponding lower and upper bound. Constraint (3.6) ensures that only p territory centers are considered. Constraints (3.7) specify that the total risk value for each territory, must not surpass its defined risk threshold. Finally, the binary nature of the variables is expressed in Constraints (3.8).

Therefore, for an instance with $|S|$ possible centers and $|B|$ basic units, the total number of decision variables is $|S||B| + |B|$.

The problem is $\mathcal{NP}$-hard. This can be argued as follows. Clearly, the problem is in $\mathcal{NP}$ because one can check feasibility of a given solution in polynomial time. Now, by reduction, if we take a special case of our problem with a very large value for parameters t_m, U_k, and β_i and a value of zero for all L_k, Constraints (3.4), (3.5), and (3.7) become redundant so we are left with a p-median problem. Thus, we have proven that the p-median problem is polynomially reducible to our problem, and since it is well-known that the p-median problem is $\mathcal{NP}$-hard, so is our problem.

Chapter 4

A Greedy Randomized Adaptive Search Procedure

In this chapter, a Greedy Randomized Adaptive Search Procedure metaheuristic is presented, which includes several well-known heuristics adapted to the problem studied in this thesis.

Although some exact optimization approaches exist for territorial design problems (Salazar-Aguilar, Ríos-Mercado, and Cabrera-Ríos (2011), Ríos-Mercado and Bard (2019), Sandoval, Díaz, and Ríos-Mercado (2020)), the vast majority of the work has been on metaheuristics, given the inherent complexity of territory design problems.

First introduced in Feo and Resende (1989) as a "probabilistic heuristic", the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic is a multiphase iterative process designed for solving combinatorial optimization problems. It combines elements of greediness and randomness to effectively navigate the solution space. The two main components of GRASP are the construction phase and the local search phase, which work to iteratively improve upon solutions.

In the construction phase, an initial feasible solution is built, starting from an empty solution and iteratively adding elements to it. At each step, a candidate

list of possible next elements is generated based on a greedy function. The key aspect of GRASP's construction phase is the incorporation of randomness: instead of choosing the best candidate (as in a purely greedy approach), one element is randomly selected from a subset of potential candidates. This subset, often referred to as the Restricted Candidate List (RCL), is formed by selecting elements that are within a certain α quality threshold or a fixed size. In this thesis, we use the first approach. The randomness introduced at this stage helps in diversifying the search, avoiding premature convergence to local optima.

Once the construction phase generates an initial solution, the local search phase seeks to improve it through an iterative process. The solution is refined by exploring its neighborhood, which is a set of solutions that can be reached through small modifications (like swapping, adding, or removing elements). The local search continues until a local optimum is found, meaning no better solution can be found in the neighborhood. This process is repeated until a stopping criterion is met, typically, a fixed number of iterations.

Since each iteration is independent, generating a different initial solution every time the construction phase is performed, which is then improved during the local search phase, GRASP has the potential to be parallelized, running each iteration on a different computer core thread. This repeated process allows the algorithm to explore various parts of the solution space, naturally balancing diversification (exploring different regions of the solution space) and intensification (searching a promising region). The randomness in the construction phase promotes diversification, while the local search phase focuses on intensification (Feo and Resende, 1995).

GRASP has been very successfully applied to a class of districting and territory design problems (Ríos-Mercado and Fernández, 2009; Fernández et al., 2010; Ríos-Mercado and Escalante, 2016) including multi-objective optimization formulations (Salazar-Aguilar, Ríos-Mercado, and González-Velarde, 2013).

In order to build a GRASP metaheuristic framework, a constructive and local

search heuristics must be developed first, to integrate them into the framework. In this chapter we define what heuristics and strategies were explored to build the proposed GRASP. The pseudocode of the GRASP is presented in Algorithm 1.

Algorithm 1 `GRASP(Instance,` $\alpha_l, \alpha_a, i_{\max}$`)`

Input: Instance = instance of the problem; α_l = RCL threshold parameter for location phase; α_a = RCL threshold parameter for allocation phase; $i_{\max}$ = maximum number of iterations.

Output: X^* allocation matrix of BUs, Y^* location vector of centers, z^* objective function value

```
1:  $X^* \leftarrow \emptyset$
2:  $Y^* \leftarrow \emptyset$
3:  $z^* \leftarrow \infty$
4:  $i \leftarrow 0$
5:  $z \leftarrow f(X, Y)$
6:  while $i < i_{\max}$ do
7:      $Y \leftarrow$ p-disp-grasp(Instance, $\alpha_l$)
8:      $X \leftarrow$ opp-cost-queue-grasp(Instance, $Y, \alpha_a$)
9:      $(X', Y') \leftarrow$ local-search(Instance, $X$, $Y$)
10:     if $f(X', Y') < z^*$ then
11:         $X^* \leftarrow X'$
12:         $Y^* \leftarrow Y'$
13:         $z^* \leftarrow z(X', Y')$
14:     end if
15:     $i \leftarrow i + 1$
16: end while
17: return $X^*, Y^*, z^*$
```

4.1 Constructive Heuristic

The foundational mathematical programming model for districting was introduced by Hess, Weaver, Siegfeldt, Whelan, and Zitlau (1965). In this paper, the *location-allocation* heuristic was first introduced. This heuristic approach breaks down the intertwined tasks of location and allocation inherent to the districting problem into two separate stages. These two stages, one focusing on the selection of territory centers and the other on the assignment of basic units to these centers, are executed in a repetitive cycle until a satisfactory outcome is achieved. The process begins with the location phase, where the centers of future territories are identified, followed by the allocation phase, which involves the distribution of basic units to the selected centers.

The constructive heuristic presented in this thesis follows the same location allocation strategy. In the location phase, the centers, or branches of the MFI, are selected to serve the BUs, while in the allocation phase, the BUs are individually allocated to an open center. Several approaches for both phases were tested and will be presented in the following sections.

4.1.1 Location Phase

Previous studies including Ríos-Mercado, Álvarez Socarrás, Castrillón, and López-Locés (2021) use the location-allocation heuristic successfully by first solving the location phase using p-partition-based heuristics to initialize the set of centroids. Another method for obtaining this initial set of territory centers is by solving the Mixed Integer Linear Programming (MILP) formulation until a feasible solution is found, however, due to the computationally complex nature of the MILP, it does not scale well for larger instances for the problem, this is shown in the next chapter. We believe it is important to once again remind the reader that classic formulations

of the TDP typically contain a single set that includes both the possible territory centers and the basic units, making them interchangeable; however, in this thesis, there are two different sets, one for centers and another one for the BUs. This adds further complexity to our problem as the heuristics used in previous works for this phase need to be adapted to our needs, more specifically the p-dispersion problem heuristic.

Two strategies were explored to use as the location phase algorithm: a p-dispersion problem heuristic and semi-linear programming. Of these, the p-dispersion problem was chosen as the algorithm to use.

4.1.1.1 The p-Dispersion Problem

Due to p being a parameter of the problem, a common strategy is to use p-partition-based heuristics to first divide the set of centers into p subsets, as stated in the previous section. Cano-Belmán et al. (2012) use the p-dispersion problem to locate the centers to be used. The p-dispersion problem, also known as the maxmin problem, aims to select a set of p points that maximizes the minimum distance between any selected pair of points in the set (Erkut and Neuman, 1991).

We are going to use the *Greedy Constructive* (GC) heuristic proposed by Erkut et al. (1994), to try to locate p territory centers from the S set, following the intuitive idea that if they are evenly dispersed, the BUs will have a close enough center to be allocated to. The use of this specific heuristic is reinforced by the contribution of Ravi et al. (1994) that states that there is no polynomial time algorithm that can provide a solution that is not worse by a factor of 2 compared to the optimal solution (unless $\mathcal{P} = \mathcal{NP}$) to the p-dispersion problem, as well as the fact that one of the conclusions of Erkut et al. (1994) is that the selected heuristic presents good enough results for problems in which p is a small proportion of all the points in the problem, which is the case for this TDP. This heuristic is presented in Algorithm 2.

Algorithm 2 GC(D', S, p)

```
Input: $D'$ distance matrix, $S$ set of nodes, $p$ number of nodes to select out of $S$
Output: $Y$ solution set $p$ centers
1: $Y \leftarrow \emptyset$
2: Find $x_1$ and $x_2$ in $S$ such that $d(x_1, x_2) = \max\{D'(x_i, x_j) : 1 \leq i < j \leq p\}$
3: $Y \leftarrow \{x_1, x_2\}$
4: while $|Y| < p$ do
5:     Find $x_i \in S \setminus Y$ such that $d(x_i, Y) = \max\{d(x, Y) : x \in S \setminus Y\}$
6:     $Y \leftarrow Y \cup \{x_i\}$
7: end while
8: return Y
```

The heuristic works by initializing an empty solution set Y on line 1, then it selects the two furthest apart points on the set S as shown on line 2, and adds them to Y.

Then it iterates until the solution set reaches p as its size, from lines 4 to 7. During each iteration, the heuristic finds the node that maximizes the minimum distance to Y and adds it to the solution set, as shown in lines 5 and 6. The distance of an individual node to the set is defined as the minimum distance of the candidate node x_i to each member of Y. Expressed in a more formal notation, the distance between a point x_i and a set Y is defined as $d(x_i, Y) = \min\{D'(x_i, x_j) : x_j \in Y, x_i \neq x_j\}$.

However, one must remember that the territorial centers have an associated T_{ik} individual business type, therefore the types must be also taken into consideration while dispersing the centers, as there is a minimum and a maximum number of centers to be used from each type.

By solving $|K|$ p-dispersion problems, considering only centers of a given $k, k \in K$ type for each problem and using $p = L_k$, we obtain an initial set of selected centers that ensures that fulfills the minimum number allowed for each k. Then, by taking

the union of the resulting centers, the Y vector is obtained as the Algorithm 3 shows. An important thing to note is that the overall p of the problem never equals the sum of all L_k, so a second phase after the union of the k sub-problems is required in order to reach the p centers required to be opened.

This phase solves yet another p-dispersion problem, taking the union of the previous centers as a starting set, and selects the most disperse node to Y, as long as the node respects the U_k constraint for its corresponding k type, until $|Y| = p$.

Algorithm 3 `p-dispersion for Y location`$(d, p, T_{ik}, L_k, U_k, S)$

Input: d distance matrix, p centers to open, T_{ik} binary matrix of center types, L_k, minimum number of centers to be used of type k, U_k maximum centers to be used of type k, S set of centers

Output: Y decision vector of located centers

```
1: Compute distance matrices, $\forall k \in K$
2: for $k \in K$ do
3:     $S_k \leftarrow \{i \in S \mid T_{ik} = 1\}$ { $S_k$ represents all centers of type $k$}
4:     $P\star \leftarrow$ GC(distance matrix of $k$, $L_k$, $S_k$) {Solve the problem for each $k$, with its corresponding nodes and distance matrix}
5:     $Y \leftarrow Y \cup \{P\star\}$ {Build $Y$ upon the partial solution}
6: end for
7: while $|Y| < p$ do
8:     for node $\in S$ do
9:         if node $\notin Y$ then
10:            Determine type of node in $S_k$
11:            Check if $U_k$ for this node type is reached; if so, skip this node
12:            Compute the minimum distance from node to all nodes in $Y$
13:            Store this minimum distance.
14:        end if
15:    end for
16:    Select the node with the maximum minimum distance
17:    Add this node to $Y$
18: end while
19: return $Y$
```

A graphical step-by-step example of the algorithm is shown in Section A.6.1.

4.1.1.2 Partial Linear Programming Relaxation

Since our problem is an $\mathcal{NP}$-hard problem according to Karp (1972), large instances are intractable by branch-and-bound methods. By introducing an integer relaxation in the X_{ij} decision variables and leaving the Y_i variables as integers, the idea is to provide to the optimization solver an easier problem to solve, meaning faster times to reach a feasible solution. The solver will provide us with valid results for Y_i, meaning the territory centers have been located. However, this solution may not represent a feasible solution of the original problem; as the X_{ij} variables are not required to be integer. But, we can use the solution values of Y as valid for the proposed heuristic. This approach is referred to as the Partial Linear Programming Relaxation (PLP).

Both of these approaches were used during the experimentation phase of the heuristic to determine which of these two strategies was the one going to be integrated in the GRASP metaheuristic. In the end, the p-dispersion problem heuristic was used as the first phase of the algorithm. The results behind this decision are detailed in Chapter 5.

4.1.2 Allocation Phase

Once the territory centers have been located, the allocation phase of our approach will allocate each individual BU to an open center. This phase is the core of the heuristic as the objective function of the problem directly measures how well clients are allocated to their nearest center whilst complying with the balancing constraints. A possible greedy approach would be to simply allocate all clients to their nearest center, but this provides infeasible solutions. Therefore, a more intelligent decision process must be introduced to achieve the maximum degree of feasibility as possible before passing the constructed solution to a repair phase.

For this phase, two algorithms are proposed, both based on the concept of

opportunity cost. The first algorithm is referred to as the Opportunity Cost with Matrix and the second one is the Opportunity Cost with Queue. The latter was chosen to be used as the allocation algorithm in the GRASP.

4.1.2.1 Opportunity Cost with Matrix

The opportunity cost refers to the value of the next best alternative that is foregone when a decision is made to choose one option over another. It represents the benefits that could have been gained by choosing an specific alternative option. This concept is crucial in economics and decision-making, as it helps in understanding the potential costs involved in not selecting the next best alternative.

Due to the constrained nature of the problem, which is limited in resources, the choice of where to allocate each BU must be made from several mutually exclusive centers. As BUs are assigned to a center, the balancing constraints of both the activities and risk measure will prevent the assignment of any more BUs to that specific center, introducing the scarcity in the mutually exclusive choices available.

By calculating the opportunity cost of all BUs as the difference of the minimal distance available to a center and all other centers, and by storing the results in a matrix with the same size as d, we can now use a heuristic that minimizes the opportunity costs across all BUs, using this matrix as the data structure which will guide the algorithm.

During the allocation phase, we are computing values for a specific i center and a specific j BU. For feasibility checks of the constructed solution, we can keep track and update only the values affected by these specific i and j. By using additional data structures, we introduce the concept of partial evaluation: instead of computing and checking the feasibility of the whole solution, we only update and check it in each individual allocation, making these computations more efficient.

For the algorithm, we start a while loop that iterates until every BU has been assigned. In this loop, we will find the indices of the n largest opportunity costs across the opportunity cost matrix, where n is a parameter of the algorithm. For each index j, which represents the BU, we find the nearest center i to this BU, taking into account that $Y_i = 1$. Afterwards, we compute and check for any violated constraint with the respective allocation of i and j. The number of violated constraints is stored in an array, from which we will pick the allocation that is feasible. If all possible allocations are infeasible, we perform another search across the matrix, checking for the next n nearest centers to the first BU in the list, as well as computing again the constraints, by eventually allocating to the center with the least amount of violated constraints. When the j BU is set to 1 in the X matrix, it is marked as assigned for the algorithm.

The time complexity for the worst case scenario of this allocation strategy is $\mathcal{O}(|B|^2 \cdot |S|)$ time, as there are constant operations of finding either the n maximum or minimum values across the whole matrix, of size $|S| \cdot |B|$.

Notice that the minimum and maximum distances do not really change over the algorithm, thus an initial precomputation of those indices may be useful to speed up times. With those insights, we present the second approach to allocate BUs.

4.1.2.2 Opportunity Cost enhanced with Data Structures

Instead of computing a matrix with all the possible costs, we only compute the opportunity cost from the best possible assignment and the worst one for each BU, storing that difference in a priority queue. A priority queue is a specialized data structure that is similar to a regular queue or stack, but where each element has a priority associated with it. In a priority queue, an element with higher priority is served before an element with lower priority. If two elements have the same priority, they are served according to their order in the queue. In this case, the opportunity

cost is the priority associated with each BU. By traversing this queue, we allocate the BUs in order of largest to least opportunity cost. The n best possible assignments for each BU do not change in distance, therefore we can sort them as well prior to the algorithm, storing the ordered centers in a dictionary. n was set to p to evaluate all the possible assignments.

Algorithm 4 defines the function that calculates such data structures, with pq being the priority queue for the opportunity cost and *best_assignments* the dictionary that stores the sorted centers nearest to each client.

Algorithm 4 `compute-structures`(d, Y, B, S, n)

```
Input: Distance matrix $d$, Decision Variable $Y$, $B$ set of BUs, $S$ set of BUs, $n$ top
    assignments
Output: Best assignments dictionary best_assignments, Priority queue pq
1: Initialize best_assignments as an empty dictionary
2: Initialize pq as a priority queue with reverse order
3: for each $j \in B$ do
4:     Initialize an array costs to store opportunity costs for $j$
5:     for $i \in Y$ do
6:         Store $d_{i,j}$ along with index $i$ in costs
7:     end for
8:     Sort costs in ascending order of cost
9:     Store the indices of the top $n$ facilities with the lowest cost in
       best_assignments[$j$]
10:    Calculate the opportunity cost for $j$ as the difference between the highest
       and lowest cost
11:    Add to the queue the $j$ BU and its opportunity cost into pq
12: end for
13: return best_assignments, pq
```

We reuse the data structures to keep track of the constraints and their status

throughout the traversal of the queue. During the queue traversal, we will mark centers as full when their respective activity measures' meet the lower bound target, forcing the algorithm to diversify and choose another center, and ensuring an even distribution among centers. Since the algorithm respects the lower bounds and it simultaneously prevents exceeding upper bounds, feasible solutions are produced more consistently than the opportunity cost with matrix algorithm in a faster time,improving the worst-case scenario from $\mathcal{O}(|B|^2 \cdot |S|)$ to $\mathcal{O}(|B| \cdot |S|)$.

However, it is still possible that a BU is left unassigned due to all of the centers being marked as full, so we must define a function to handle unassigned BUs. This is a very simple function that will simply assign those BUs to the center that has the most capacity left in the β risk threshold.

This algorithm is better in every way than the previous approach of the opportunity cost stored in a matrix, due to the use of data structures and a better understanding of the problem and exploitation of its underlying structure. A graphical step-by-step example of this algorithm is shown in Section A.6.2

4.2 GRASP Metaheuristic Adaptations

The proposed GRASP metaheuristic required an adaptation of the mentioned algorithms to introduce the concept of the Restricted Candidate List (RCL). In this case, we use a value-limited RCL using two α parameters: α_l for the location phase and α_a for the allocation phase. In Chapter 5, we present the results that guide our decision towards using the p-dispersion strategy for the location phase and the opportunity cost enhanced with the queue and other data structures as the allocation phase. In this section, we present the corresponding modifications in order to introduce the RCL in our GRASP.

Algorithm 5 implements a GRASP (Greedy Randomized Adaptive Search Procedure) heuristic for the p-dispersion problem. The procedure is as follows:

Taking as input an instance of the problem comprising a distance matrix d, a set of points S, the number of centers p to select, and a quality threshold α_l for constructing the Restricted Candidate List (RCL), the algorithm identifies the pair $(x_i, x_j) \in S$ that maximizes d_{x_i,x_j}. These points form the initial solution set $Y = x_i, x_j$.

In order to iteratively construct the solution, the algorithm iterates while $|Y| < p$, using the greedy function $\phi : S \setminus Y \to \mathbb{R}$ as $\phi(x_i) = \min\{d_{x_i,x_j} : x_j \in Y\}$, which computes the minimum distance from a candidate point to the current solution set, and determines $\phi_{\max} = \max\{\phi(x_i) : v \in S \setminus Y\}$ and $\phi_{\min} = \min\{\phi(x_i) : x_i \in S \setminus Y\}$. With these two values, it constructs the RCL as
RCL: $\{x_i \in S \setminus Y : \phi(x_i) \geq \phi_{\max} - \alpha_l(\phi_{\max} - \phi_{\min})\}$. This set comprises candidate points that are relatively distant from the current solution set, and finally it randomly selects a point $x_i \in$ RCL and updates $Y = Y \cup \{x_i\}$.

This approach employs a balance between greedy selection and randomization. The greedy component favors points that maximize the minimum distance to the current solution set, while the randomized component introduces diversity by selecting from a set of good candidates. The parameter $\alpha_l \in [0, 1]$ controls the restrictiveness of the RCL: as α_l goes to 0, the algorithm becomes more greedy, whereas larger values of α_l introduce more randomness into the selection process.

It is important to notice that this algorithm is used to solve the k sub-problems for each center type. Once the k solutions have been joined into the larger set, it reuses the same ϕ greedy function, adding centers to the RCL, as long as the candidate center respects the U_k constraint of its corresponding k type.

Algorithm 5 `p-dispersion-grasp`(Instance, α_l)

```
Input: Instance of the problem, $\alpha_l$ quality threshold for the distance
Output: $Y$ solution set of p centers
1: $d$, $S$, $p \leftarrow$ parameters from the Instance.
2: $N \leftarrow |S|$
3: Find $(x_i, x_j)$ such that $d_{x_i,x_j} = \max\{d_{x_i,x_j} : 1 \leq 1 < j \leq |S|\}$
4: $Y \leftarrow \{x_i, x_j\}$
5: while $|Y| < p$ do
6:     Define $\phi(x_i) = \min\{d_{x_i,x_j} : x_j \in Y\}$ for $x_i \in \{1, \dots, N\} \setminus Y$
7:     $\phi_{\max} \leftarrow \max\{\phi(x_i) : x_i \in \{1, \dots, N\} \setminus Y\}$
8:     $\phi_{\min} \leftarrow \min\{\phi(x_i) : x_i \in \{1, \dots, N\} \setminus Y\}$
9:     $RCL \leftarrow \{x_i \in \{1, \dots, N\} \setminus Y : \phi(x_i) \geq \phi_{\max} - \alpha_l(\phi_{\max} - \phi_{\min})\}$
10:    Select random $x_i$ from RCL
11:    $Y \leftarrow Y \cup \{x_i\}$
12: end while
13: return $Y$
```

On the other hand, for the allocation phase, instead of choosing the nearest assignment for the BUs in the priority queue, we define a greedy function $\phi(j)$: $\min(d_{i,j}, i \in Y)$ that minimizes the distance to a BU j from all available centers in Y, defining $\phi_{\max}$ as the furthest center to j and $\phi_{\min}$ as the nearest center to j. In this case, due to using precomputed data structures and sorting the centers, $\phi_{\max}$ is the last element in the dictionary values for the best assignments of a BU j, and $\phi_{\min}$ is the first element in such values, as they are sorted already. The RCL is constructed as RCL $\leftarrow \{j : \phi(j) \leq \phi_{\min} - \alpha_a(\phi_{\max} - \phi_{\min})\}$. This is detailed in Algorithm 6.

Algorithm 6 `opp-cost-queue-grasp`(Instance, Y, α_a)

```
Input: Instance of the problem, $Y$ decision variable, $\alpha_a$: quality threshold for the
    allocation RCL
Output: $X$ decision matrix of allocations
 1: Distance matrix $d$, Decision Vector $Y$, $B$ set of BUs, $S$ set of BUs, $p$ centers to
    be used , $\beta$ risk threshold $\leftarrow$ Instance parameters
 2: $pq$, $best\_assignments \leftarrow$ compute-structures$(d, Y, B, S, p)$
 3: $assigned\_clients \leftarrow \emptyset$
 4: $full\_centers \leftarrow \emptyset$
 5: while $pq$ is not empty and $|assigned\_clients| < B$ do
 6:     $j \leftarrow$ dequeue $pq$
 7:     if $j \notin assigned\_clients$ then
 8:         $\phi_{\max} \leftarrow$ last element from $best\_assignments[j]$
 9:         $\phi_{\min} \leftarrow$ first element from $best\_assignments[j]$
10:         RCL$\leftarrow \{j : \phi(j) \leq \phi_{min} - \alpha_a(\phi_{max} - \phi_{min})\}$
11:         Select random $i$ from RCL as long as it is not in $full\_centers$
12:         $X_{i,j} \leftarrow 1$
13:         Add client to $assigned\_clients$
14:         Update $values\_m$, $risk\_v$ with $i, j$
15:         if center $i$ has reached lower bounds in activity measures or in the risk
            value then
16:             Add $i$ to $full\_centers$
17:             break
18:         end if
19:     end if
20: end while
21: Assign all clients $\notin assigned\_clients$ to the centers with the largest capacity
    available in the risk threshold $\beta$.
22: return $X$
```

4.3 Local Search Heuristics

Local search heuristics operate on the principle of iteratively exploring the neighborhood of a current solution in the search space to find a new solution that is better according to a given objective function (Taillard, 2023). *The neighborhood* of a solution typically consists of all solutions that have a small, predefined change in their structure from the current solution. The steps taken to move from one solution to another are defined by what is called a *move operator*.

They explore the local space of a given initial solution trying to either improve the objective function value or repair its feasibility of the solution, we two strategies were implemented and tested: (i) A Best Found (BF) strategy that moved to the best improving neighbor and (ii) a First Found (FF) strategy that moved to the first improving neighbor. In our testings, shown in Table 5.4, the FF yield better results, so this is the strategy chosen for the final GRASP implementation.

In this work, some infeasible initial solutions can be a result of the constructive heuristic, because of this, we have developed a Local Search procedure that can handle infeasibility by first repairing the solutions, once solutions are feasible, it moves on improving the solution quality. This procedure is explained below.

4.3.1 Repairing Infeasibilities

As stated previously, the constructive heuristic proposed in this thesis may provide solutions that are not feasible. A solution may be infeasible because of two reasons: a center exceeds the maximum number of allocated BUs, because of the activities' measures surpassing their maximum target, or the risk threshold is exceeded; or the BUs assigned were not enough to complete the minimum required in the activities' measures. To avoid this, the local search applies one of the two following basic moves: **Add**(i) which will add BUs to a center i, that is violating the lower bounds

of the constraints until those bounds are met, while ensuring that the moved BUs do not render their previous centers infeasible, and **Remove**(i) which will remove BUs from a center i that exceeds the upper bounds of the constraints, ensuring that whichever center those BUs end up being reallocated to does not become infeasible as well.

By directing the local search with "AddTo" or "RemoveFrom" sets for centers, we achieve faster computations, as it is a straight-forward to determine to which set does a center is member of, as shown in Algorithm 7. The moves are applied until all centers are feasible, transforming the solution to a feasible one, expressed in Algorithm 8.

Algorithm 7 `compute-add-remove`(Instance, X, Y)

```
Input: Instance of the problem, $X$ decision matrix, $Y$ decision vector
Output: $AddTo$ set of centers to add BUs, $RemoveFrom$ set of centers to remove
    BUs
1: for $i \in Y$ do
2:     if Any value from $X, Y_i$ is greater than the upper bounds from the con-
       straints then
3:         $RemoveFrom \leftarrow i$
4:     else if Any value from $X, Y_i$ is less than the lower bounds from the con-
       straints then
5:         $AddTo \leftarrow i$
6:     end if
7: end for
8: return $AddTo, RemoveFrom$
```

Algorithm 8 `repair-solution`(Instance, X, Y)

```
Input: Instance of the problem, $X$ decision matrix, $Y$ decision vector
Output: $X'$ decision matrix
1:  $AddTo, RemoveFrom \leftarrow$ compute-add-remove(Instance, $X$, $Y$)
2:  $X'$ copies $X$
3:  for $\tilde{i} \in AddTo$ do
4:      while Any value from $X, Y_{\tilde{i}}$ is less than the lower bounds from the constraints do
5:          Allocate the nearest $j$ BU to $\tilde{i}$, removing it from its previous center $i*$
6:          $X'_{i*,j} = 0$
7:          $X'_{\tilde{i},j} = 1$
8:          Update the constraints' states for $\tilde{i}$ and $i*$
9:      end while
10: end for
11: for $\tilde{i} \in RemoveFrom$ do
12:     while Any value from $X, Y_{\tilde{i}}$ is greater than the upper bounds from the constraints do
13:         Move the furthest $j$ BU from $\tilde{i}$, allocating it to its nearest center $i*$
14:         $X'_{\tilde{i},j} = 0$
15:         $X'_{i*,j} = 1$
16:         Update the constraints' states for $\tilde{i}$ and $i*$
17:     end while
18: end for
19: return $X'$
```

4.3.2 Improving objective function value

Once the solution is feasible, the local search can now proceed to improve the objective function value. For this, the following moves are defined:

- **Allocate Other**(i, j): Reassign a BU j from center i to another center $\tilde{i}$, where $\tilde{i} \neq i$, $\iff X_{ij} = 1$.
- **Interchange**$(i, j, \tilde{i}, \tilde{j})$: Swap allocations of BUs j and $\tilde{j}$ such that j is allocated to $\tilde{i}$ and $\tilde{j}$ to i, $\iff X_{ij} = 1 \land X_{\tilde{i}\tilde{j}} = 1$.
- **Deactivate**$(i, \tilde{i})$: Close center i and open an unused center $\tilde{i}$. Assign BUs to $\tilde{i}$ until the lower bounds of the balancing constraints are met, and reallocate the remaining BUs from i to the best-fit centers, $\iff Y_i = 1 \land Y_{\tilde{i}} = 0$.

The Deactivate move reallocates the remaining BUs using the same dictionary with the best assignments to each BU from the allocation phase of the constructive heuristic, as it already has the centers sorted by their distance to the BUs.

4.3.3 Combining Multiple Neighborhoods

As we have designed multiple neighborhoods for the local search phase, it is natural to combine these multiple moves in a cyclical procedure, exploring the local space of a move until it reaches the local optimum value and then it moves on to the next move, until no move improves the solution. This helps us escape local optima and provides a more comprehensive exploration of the solution space. The main steps of the algorithm are are:

1. Start with an Initial Solution: Begin with an initial solution and a set of predefined neighborhood structures. In this case, the initial solution is the solution provided by the constructive heuristic and the set of neighborhoods is the collection of the 3 moves defined previously.

2. Sequential Neighborhood Search: Explore these neighborhoods in a systematic way. For each neighborhood, perform a local search to find a local optimum. In this case, we performed the **Allocate to Other, Interchange** and **Deactivate** moves sequentially in that order, applying the moves iteratively to the result of the

previous move until all three neighborhoods reached their local optimum, doing so cyclically.

3. Neighborhood Change: If an improvement is found in any neighborhood, the process is restarted from the first neighborhood with the improved solution. If no improvement is found in any of the neighborhoods, the process stops, and the best-found solution is returned.

In Algorithm 9, lines 1 to 3 repair the solution using the previously defined algorithms. The cyclical strategy is applied as a while loop from lines 6 to 29, applying the moves in the sequential order show, until no move improves the incumbent solution. A solution is expressed as (X, Y) where X is the decision variable matrix of allocations and Y the decision variable vector of locations.

Algorithm 9 `local-search`(Instance, X, Y)

Input: Instance of the problem, X decision matrix, Y decision vector

Output: X decision matrix, Y decision vector

```
1: if $(X, Y)$ is not feasible then
2:     $X \leftarrow$ repair-solution($(X, Y)$)
3: end if
4: improvement $\leftarrow$ True
5: $z \leftarrow f(X, Y)$
6: while improvement do
7:     improvement $=$ False
8:     $(X', Y') \leftarrow$ Apply Allocate Other to $(X, Y)$ until local optimum is met
9:     $z' \leftarrow f(X', Y')$
10:    if $z' < z$ then
11:        $(X, Y) \leftarrow (X', Y')$
12:        $z = z'$
13:        improvement $=$ True
14:    end if
15:    $(X', Y') \leftarrow$ Apply Interchange to $(X, Y)$ until local optimum is met
16:    $z' \leftarrow f(X', Y')$
17:    if $z' < z$ then
18:        $(X, Y) \leftarrow (X', Y')$
19:        $z = z'$
20:        improvement $=$ True
21:    end if
22:    $(X', Y') \leftarrow$ Apply Deactivate to $(X, Y)$ until local optimum is met
23:    $z' \leftarrow f(X', Y')$
24:    if $z' < z$ then
25:        $(X, Y) \leftarrow (X', Y')$
26:        $z = z'$
27:        improvement $=$ True
28:    end if
29: end while
30: return $(X, Y)$
```

4.4 Parallel computing

As the individual components for the heuristics were chosen due to their performance detailed in Chapter 5, the GRASP metaheuristic was built from these individual algorithms. An advantage of the GRASP metaheuristic is that each individual iteration can be parallelized, as each iteration is unique and independent from the next or previous one. By assigning an iteration to a computer core thread, we can speed up the computation by introducing parallelization with the use of such computer threads, synchronizing the threads only when checking for the best solution found across the multi-threaded computation. This parallelization mechanism was successfully implemented for the GRASP metaheuristic, providing faster results compared to a single-threaded GRASP version, as shown in section 5.5.1.1.

In order to properly implement a multi-threaded version of the GRASP metaheuristic we had to use the computer science concept of locks. A lock, also known as a mutex (derived from mutual exclusion) is a construct that allows us to safely manage different state variables, by not allowing them to be accessed from multiple threads at the same time, preventing race conditions, which occur when two different threads try to modify the same variable at the same time, entering into a “race” of accessing and modifying the variable, ensuring a safe and correct implementation of the algorithm.

Chapter 5

Computational Results

In this chapter we present the results of the computational experience of the proposed algorithm. We highlight the performance advantages of using data structures and an intelligent design of algorithms versus a more naive version of the algorithm. Additionally, we compare the obtained solutions with those provided by Gurobi version 11.0.3, a state-of-the-art solver. We also show the effectiveness of implementing a parallel version of GRASP, reducing running times significantly. Finally, we provide the best parameters to be used in the GRASP metaheuristics as result of a fine-tuning phase.

5.1 Instance generation

To assess our procedure, some problem instances were randomly generated. Throughout the thesis, the Julia programming language is used to carry out all the computations related to the problem. The Julia language was designed to be used in scientific computing contexts, with a simple syntax reminiscent of Python and FORTRAN while providing the performance of C/C++, showing that one can have machine performance without sacrificing readability (Bezanson et al., 2014). From the instance generation, to the modeling of the problem as a mathematical program and the GRASP metaheuristic, every stage of computation was written in Julia.

In order to generate problem instances, several routines are detailed and executed in the following order: first, the coordinates in a Cartesian plane of the BUs and the centers are randomly generated from an interval of 5 to 10,000; second, the d distance matrix is calculated from these coordinates using the Euclidean distance metric, rounding the distances to integers. Third, the T_{ik} parameter is generated, assigning a random k type to each center, we define the L_k and U_k bounds proportional to the the sum of all the centers for each type k. Afterwards, for each activity measure m, a random value is assigned to every individual BU j in the vector v_m^j, from a range of 10 to 30. Once all the BUs have an activity measure value for all the activities, the targets of the measures are calculated as the sum of all the measure values divided by $|S|$ and multiplied by an additional τ parameter to introduce some leeway, with a value of 1. The t_m tolerance parameter is defined as 0.4. Finally, the risk value for every BU j is randomly generated in the 40 to 60 range, with the risk threshold for each center i defined as the sum of all the individual values divided by $|S|$ and multiplied by the same τ parameter $((\sum_{j \in B} \frac{r_j}{S})\tau)$.

All of the instances are written using an HDF5 standard compliant file format, in order to allow serialization and data sharing between programming languages and computing platforms, as long as the programming language has a valid HDF5 parser library to read the instances and the solutions (Koranne, 2011).

Three sizes of instances were generated, the BUs and centers' coordinates were generated from a range of [5, 10,000] in the Cartesian plane, with the following configuration:

TABLE 5.1: Sizes of the generated instances

Size	$\|B\|$	$\|S\|$	p	Number of instances
1	625	62	32	20
2	1250	155	62	20
3	2500	325	125	10

5.2 Exact solutions

Algebraic modeling languages (AMLs) are domain specific computer languages that are used to describe and solve mathematical optimization models with a similar syntax to the mathematical notation of optimization problems. This makes it possible to define optimization problems in a very clear and understandable manner. This is facilitated by specific language components such as sets, indices, algebraic expressions, sparse indices, variables for handling data, and constraints with arbitrary names (Kallrath, 2004).

Instead of directly solving those problems, an AML calls relevant external algorithms to optimize the problem. These programs, known as solvers, use exact algorithms designed to handle a wide range of optimization problems, such as linear programs, integer programs, and quadratic programs, for example. The algorithms use mainly branch and bound, and algorithmic strategies depending on the structure of the structure of the problem and the solver itself. These algorithms can also be used to implement more advanced methods such as branch and cut, branch and price, or cutting planes.

By using an AML to represent the mathematical model described in Chapter 3, we can feed the resulting abstraction of the model to an optimization software package.

In this specific case, the algebraic modeling language used is the JuMP.jl package from the Julia language. JuMP enables developers to use a very concise syntax that enables a near one-to-one mapping of the mathematical notation to the language features (Dunning et al., 2017), and the mathematical solver used is the Gurobi Optimizer, a commercial solver that uses state-of-the-art optimization algorithms and techniques to solve problems, providing better results than open source solvers can (Gurobi Optimization, LLC, 2023). We briefly tested the CPLEX and HiGHS optimizers but decided to use Gurobi based on the preliminary results when comparing

to the mentioned solvers.

Using the Julia language version 1.10.4 and Gurobi version 11.0.3, we modeled the problem using JuMP version 1.22.2 and introduced the corresponding model and instance to the solver, setting two time limit for the optimization process of 1800 seconds and 7200 seconds, varying on the size of the instance. The results reported are the total optimization time of the algorithm, the best integer solution found, the best lower bound, as well as the relative optimality gap or MIP (Mixed Integer Programming) gap. According to Gurobi Optimization, LLC (2023), "the MIP gap is defined as $|z_P - z_D|/|z_P$, where z_P is the primal objective bound (i.e., the incumbent objective value, which is the upper bound for minimization problems), and z_D is the dual objective bound (the lower bound for minimization problems)". The MIP gap for the termination of Gurobi's algorithm is left as the default value of 0.01%, a hundredth part of 1%.

For Size 1, the results are shown in Table 5.2. As appreciated, out of the 20 instances, no solutions were proven to be optimal within the time limit of 1800 seconds (half an hour). The average relative optimality gap (MIP gap) percentage of the solutions is 0.28%, proving that even for the smallest instances, the $\mathcal{NP}$ nature of the problem is impacting the performance of exact solvers, although the gap to optimality is minimal. This is useful when comparing to the results of the heuristics, as we can compare to near optimal results provided by the best bound.

As shown in the Appendix in tables A.1 and A.3, for Sizes 2 and 3, no optimal solution was found within half and hour and 2 hours, the respective time limits for the instances. The average MIP gap percentage of Size 2 is 0.45% and for Size 3 is 0.18%. These values represent almost optimality, in most cases, the optimal solution was reached and the algorithm spent the rest of the time proving optimality. Since we are dealing with an $\mathcal{NP}$-hard problem, Gurobi struggles to prove that the best solution is the optimal one, spending computational time, showcasing the usefulness of the heuristics.

Table 5.2: Exact solver solutions for Size 1

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	1800	464274	462154.44	0.46
2	1800	475106	474267.15	0.18
3	1800	451907	451740.15	0.04
4	1800	460877	460520.42	0.08
5	1800	469190	468298.94	0.19
6	1800	491611	490305.18	0.27
7	1800	454550	453366.41	0.26
8	1800	462113	460970.18	0.25
9	1800	488971	487836.91	0.23
10	1800	471810	467610.27	0.89
11	1800	468195	466413.29	0.38
12	1800	461823	461040.14	0.17
13	1800	468480	468091.44	0.08
14	1800	459301	459097.84	0.04
15	1800	483950	479672.39	0.88
16	1800	462365	461967.27	0.09
17	1800	462745	462126.53	0.13
18	1800	455692	454556.37	0.25
19	1800	461881	458774.94	0.67
20	1800	465070	464547.88	0.11

5.3 Constructive heuristics

In this section the results of the different strategies that compose the constructive heuristic are presented. As a reminder, the heuristic has two main phases: location and allocation. For location, we defined that we can use either a p-dispersion based heuristic or the Partial Linear Programming (PLP henceforth) relaxation approach. For allocation, we can use the opportunity cost algorithm that calculates the whole opportunity cost matrix or the algorithm that incorporates data structures such as the opportunity cost priority queue, referring to each as "matrix" or "queue" algorithms, respectively. All instances of Sizes 1 and 2 were solved with all the possible combinations of location and allocation strategies. In Table 5.3 we can see the averages of the run-time of the algorithms and the number of constraints violated by the resulting solutions, which gives us an insight on how far away are the solutions from feasibility. ε denotes a time equal or shorter than 0.01 seconds. The Total Time column is the sum of the location and allocation phases.

TABLE 5.3: Comparison among the Constructive Heuristic strategies (averages)

Inst. Size	Location Strat.	Location Time (s)	Allocation Time (s)		Total Time (s)		# of violated constraints before repair	
			Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.
1	P-disp	0.02	0.22	ε	0.24	0.04	43.75	5.4
	Relaxation	1.00	0.31	ε	1.31	1.01	49.75	4.6
2	P-disp	0.03	1.60	0.02	1.63	0.05	81.55	0.15
	Relaxation	11.61	1.63	0.02	13.23	11.79	92.01	0.30

The p-dispersion location strategy outperforms the partial linear programming relaxation strategy in execution time, therefore we will use it as the building block for our GRASP metaheuristic. Meanwhile, the queue allocation strategy consistently outperforms the matrix allocation strategy in both run time and feasibility of the created solution. As seen in the table, as the instances get larger, the queue allocation algorithm almost always provides feasible solutions, reducing the need of using the

repair algorithm in the local search procedure.

5.4 Local search heuristics

As defined in the previous chapter, there are three moves to perform in the local search procedure: **Allocate to Other, Interchange** and **Deactivate**. These three moves were used together in a cyclic local search, in the same order, applying the moves iteratively to the result of the previous move until all three neighborhoods reached their local optimum. Local search heuristics can guide their search by either moving to the best solution found across the neighborhood or move to the first solution that improves the objective function value, without exploring all the alternatives. We define these two strategies as Best Found (BF) or First Found (FF). Applying the local search with either BF or FF strategy to the solutions produced by the constructive heuristic we present the following results:

Table 5.4: Evaluation of local search strategies (averages)

Inst. Size	Strategy	% Rel. Optimality Gap	Time (s)
1	BF	10.88	0.89
	FF	15.49	0.52
2	BF	10.73	10.98
	FF	10.52	6.37

As appreciated, the Best Found strategy provides slightly better results on average in the optimality gap to the best bound in the Size 1 instances, but requires a longer time. For Size 2, the gaps are comparable, and times are shorter for First Found. Therefore, we chose the First Found strategy to be used in the GRASP search procedure, preferring shorter run-times.

5.5 GRASP

With the p-dispersion and opportunity cost queue enhanced strategies chosen for our constructive heuristic and the First Found strategy along with the cyclical mechanism for the search heuristic, we can now integrate and adapt these building blocks into the Greedy Randomized Adaptive Search Procedure metaheuristic, defining their respective modifications in Chapter 4.

GRASP requires the α_l and α_a quality threshold parameters for the RCL as well as the number of iterations that it will run for. Moreover, as our GRASP can be parallelized, the number of computer core threads used to parallelize the computation needs to be calibrated as well for maximum performance.

5.5.1 Fine-tuning of GRASP parameters

5.5.1.1 Number of threads

The first parameter that needs to be calibrated is the number of threads that the GRASP metaheuristic will run on. We defined a basic configuration with $\alpha_l = 0.1, \alpha_a = 0.1$ and 50 iterations. As the number of threads depends on the number of cores of each computer's CPU, we decided to run from 1 to 16 threads, as that is the maximum number of threads in the computing platform in which we ran the experiments. We obtained the results shown in Figures 5.1 and 5.2.

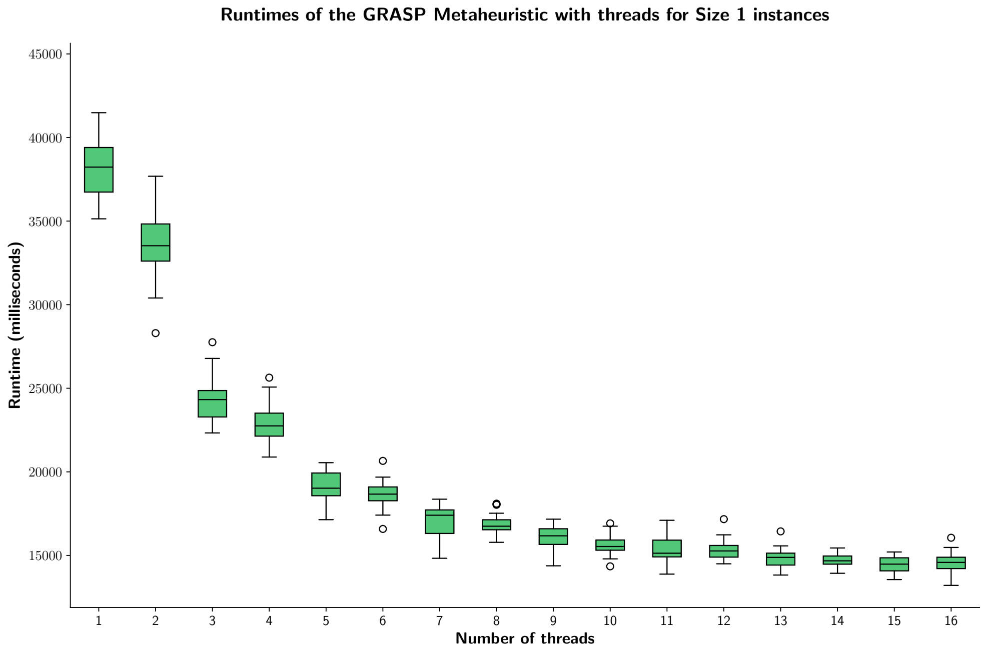


Figure 5.1: Boxplot of the GRASP's run-time for Size 1 instances

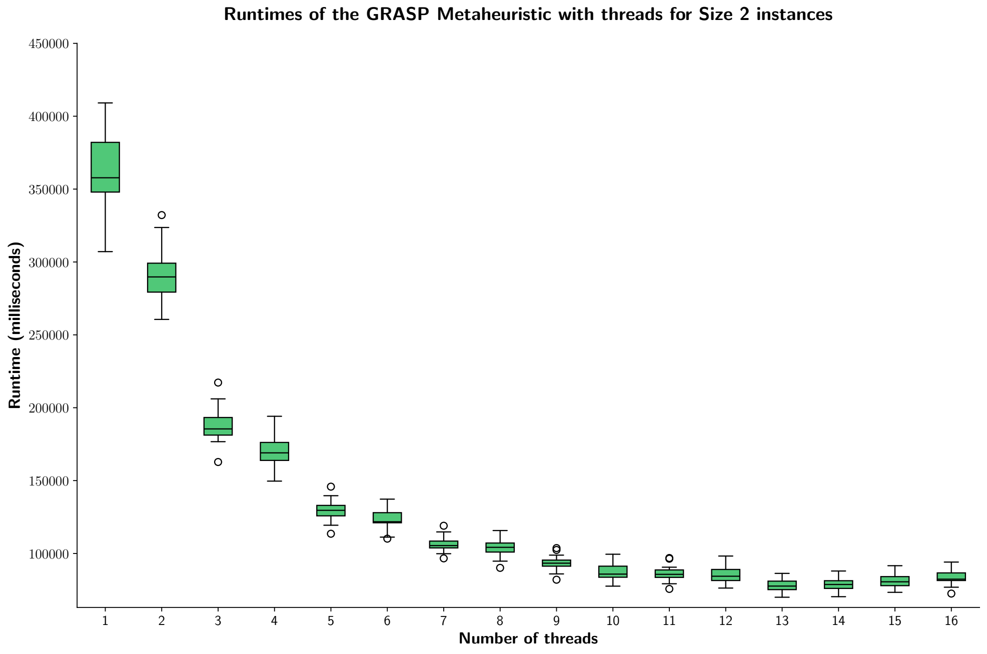


Figure 5.2: Boxplot of the GRASP's run-time for Size 2 instances

Statistical hypothesis testing was performed on the results to determine the optimal amount of threads to use as well as to check if that number of threads had any impact on the objective function value, which is something undesired.

The full extent of the statistical tests is present in the Appendix, more specifically section A.5, the main conclusions extracted were that: increasing the number of threads generally improves performance, with the most substantial gains observed when moving from 1 to 14 threads; the number of threads for minimizing run-time appears to be around 14 or 15; performance plateaus or shows minor reductions after 14 threads, increasing more the number of threads does not have a significant impact on the objective value, as evidenced by the ANOVA (Analysis of Variance) on objective values. After 14 threads, the overhead of communication between the computing processes, the scheduler of the operating system and the memory requirements negate any further improvement in the computation time of the heuristic.

A very interesting thing to note as well is that the size of the problem impacts the performance per thread. For Size 1, we minimize run-times using 16 threads, but due to a heavier computing load for Size 2, we find a better run-time with 14 rather than 16 threads. This can be explained using Amdahl's law (Amdahl, 1967), a computer architecture law that is used in parallel computing to explain and predict the speedup of a process if ran in multiple processors or cores. A heavier load may also impact the task scheduler of the host operating system, larger instances may also incur more memory costs, having a larger overhead, and in this case, making the increasing thread count a penalty from 15 threads onward.

When testing for Size 3 instances, a high variability when running different numbers of threads was found. In some runs, 8 threads provided the best times, whereas for the same instance, if ran with 12 threads at a later point in time, it ran faster than 8 threads, but when trying to replicate even later, the 12 threads were worse than the 8 threads. This was a hindrance on testing and calibrating the number of threads for larger instances, but was an interesting finding, as it confirms

that the size of the instance does impact the best number of threads, pointing to the computing overhead of more complex instances. The reasons of this inconsistent behavior were not made clear, so future work could tackle this issue.

It is important to note that Gurobi will use all available CPU cores by default, making it parallel as well.

5.5.1.2 Calibration of α parameters

Next we have to calibrate the α parameters as well as the number of iterations. We tested the following combinations for all instances of Sizes 1 and 2: $\alpha_l \in \{0.1, 0.3, 0.5, 0.7\}$, $\alpha_a \in \{0.1, 0.3, 0.5, 0.7\}$ and *iters* $\in \{25, 50, 75, 100\}$, giving us a total of 64 combinations to run. In Tables A.28 and A.29 we present the statistical description of running the GRASP for all the combinations, solving 10 instances of Sizes 1 and 2 each.

We carried out one-way Analysis of Variance (ANOVA) on the three parameters and whether they have an impact or not in the run-time of the metaheuristic as well as the objective function value. For the iterations parameter, the ANOVA test on both results revealed statistically significant differences across the different levels of iterations, meanwhile both α did show statistically significant differences across their different levels for instances of Size 1, but the same behavior was not seen in instances of Size 2. We also carried out a Factorial ANOVA to statistically control for variables and the interactions they may have between each other, obtaining varied results for both instance sizes. As the number of iterations increases, so does the run-time and the objective function improves slightly, but the results for α are inconclusive, as different results are observed across the different sizes, they may also be due to the factorial nature of this test. Finally, we performed Tukey's Honest Significant Differences testing, showing that α_l and α_a have impact on the objective function value for instances of Size 1, but not for instances of Size 2. Therefore, we decided

to settle for $\alpha_l = 0.1, \alpha_a = 0.3$ and $iters = 100$. Both α have good enough averages across all of the combinations for the two instance sizes and the number of iterations decreases our relative optimality gap, incurring a time penalty that is acceptable when comparing to the run-time of Gurobi.

5.5.2 GRASP performance

With $\alpha_l = 0.1, \alpha_a = 0.3$, $iters = 100$ and the number of threads to be used set to 14, we can now properly run the GRASP metaheuristic and obtain results for all of the instances. In this final experiment, we also used 10 instances of Size 3 to evaluate both the metaheuristic as well as Gurobi to their limit. Due to thermal throttling and inconsistent core usage, we had to reduce the number of iterations to 25 in the instances of Size 3. In this case, Gurobi was set a time limit of two hours, and the results can be found in the Appendix, on Table A.3

The most important table that can condense the entire work of this thesis is the following:

Table 5.5: Results of the Fine-tuned GRASP with 14 threads

Instance Size	% Optimality Gap			Speedup Factor vs Gurobi		
	Min	Avg	Max	Min	Avg	Max
1	5.22	6.71	8.56	82.68	124.74	144
2	6.33	7.55	9.91	11.05	12.25	13.45
3	7.38	8.34	8.97	3.84	4.26	4.95

The optimality gap to Gurobi's best bound z_D is calculated with the same formula defined by Gurobi Optimization, LLC (2023), which in this case z_P is the value provided by the GRASP metaheuristic.

As the table shows, the GRASP metaheuristic runs several orders of magnitude

faster than Gurobi while providing solutions that are a single digit of relative difference with respect to the solutions found by Gurobi. For all the tables of results, when comparing to Gurobi, the relative difference is how worse are the solutions found by the heuristics.

For Instance Size 1, GRASP runs more than 100 times faster than Gurobi, providing solutions that have a 6.71% of optimality gap.

For Instance Size 2, the speedup is not as dramatic but it is still on average 12 times faster, with a slightly worse average optimality gap of 7.55%.

Finally, for Instance Size 3, the speedups are still there, reduced, but still 4 to 5 times faster than Gurobi, with the optimality gap still a digit away. In this final instance size, Gurobi had several out-of-memory errors when working with the instances, so multiple retries had to be done to solve the problem until it reached the specified timeout.

The full results are present in the following tables:

Table 5.6: GRASP results on Size 1 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	1800	464274	0.46	462154.4	15.32	501167	7.78	117.49
2	1800	475106	0.18	474267.2	13.33	507463	6.54	135.03
3	1483	451907	0.04	451740.2	13.48	491261	8.04	133.53
4	1800	460877	0.08	460520.4	13.63	498939	7.70	132.06
5	1800	469190	0.19	468298.9	14.24	496673	5.71	126.40
6	1800	491611	0.27	490305.2	14.90	522147	6.10	120.81
7	1800	454550	0.26	453366.4	12.50	495818	8.56	144.00
8	1800	462113	0.25	460970.2	12.54	497735	7.39	143.54
9	1800	488971	0.23	487836.9	13.23	517519	5.74	136.05
10	1800	471810	0.89	467610.3	21.77	506917	7.75	82.68
11	1800	468195	0.38	466413.3	14.86	507179	8.04	121.13
12	1800	461823	0.17	461040.1	14.91	489697	5.85	120.72
13	1800	468480	0.08	468091.4	13.93	499595	6.31	129.22
14	976	459301	0.04	459097.8	14.11	499018	8.00	127.57
15	1800	483950	0.88	479672.4	14.35	506120	5.23	125.44
16	1384	462365	0.09	461967.3	13.12	496302	6.92	137.20
17	1800	462745	0.13	462126.5	14.88	492972	6.26	120.97
18	1800	455692	0.25	454556.4	14.11	480927	5.48	127.57
19	1800	461881	0.67	458774.9	14.51	485495	5.50	124.05
20	1800	465070	0.11	464547.9	14.88	490130	5.22	120.97

Table 5.7: GRASP results on Size 2 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	1800	949005	0.44	944849.5	145.24	1022786	7.62	12.39
2	1800	975979	0.92	967012.3	140.51	1043541	7.33	12.81
3	1800	988670	0.67	982048.4	159.46	1082240	9.26	11.29
4	1800	974186	0.29	971396.9	147.49	1049803	7.47	12.20
5	1800	958455	0.05	957950.0	135.87	1039603	7.85	13.25
6	1800	964088	0.22	961976.2	139.55	1036826	7.22	12.90
7	1800	968567	0.56	963149.9	143.17	1039149	7.31	12.57
8	1800	986315	0.32	983207.5	133.83	1049604	6.33	13.45
9	1800	951731	0.45	947419.2	139.09	1025924	7.65	12.94
10	1800	999440	0.84	991050.3	159.94	1100104	9.91	11.25
11	1800	961532	0.70	954783.3	137.66	1022340	6.61	13.08
12	1800	942864	0.20	940956.3	152.63	1008988	6.74	11.79
13	1800	1012710	0.32	1009464.0	162.79	1114006	9.38	11.06
14	1800	987762	0.28	985021.3	153.46	1058794	6.97	11.73
15	1800	961675	0.48	957101.4	148.60	1030436	7.12	12.11
16	1800	990828	0.70	983915.7	160.70	1057756	6.98	11.20
17	1800	992812	0.13	991539.1	145.73	1067347	7.10	12.35
18	1800	980761	0.64	974474.1	141.49	1056380	7.75	12.72
19	1800	973562	0.28	970867.4	136.73	1040261	6.67	13.16
20	1800	972605	0.45	968213.7	154.64	1049519	7.75	11.64

Table 5.8: GRASP results on Size 3 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	7200	883693	0.29	881117.8	1746.70	959509	8.17	4.12
2	7200	893408	0.15	892105.5	1655.75	963215	7.38	4.35
3	7200	882602	0.04	882275.8	1452.81	966988	8.76	4.96
4	7200	913347	0.26	910961.7	1748.65	1000684	8.97	4.12
5	7200	895581	0.21	893657.9	1633.76	975003	8.34	4.41
6	7200	896157	0.18	894518.5	1873.31	966452	7.44	3.84
7	7200	893684	0.13	892487.9	1626.62	979920	8.92	4.43
8	7200	895125	0.28	892662.6	1641.34	974882	8.43	4.39
9	7200	877510	0.12	876485.6	1836.52	957017	8.41	3.92
10	7200	892898	0.14	891652.1	1657.39	975116	8.56	4.34

Chapter 6

Conclusions

6.1 Main findings and conclusions

By using intelligent algorithms as well as efficient data structures, we solved the Territorial Design Problem for microfinancial institutions using a GRASP metaheuristic that proved to be competitive with commercial exact optimization software, being several orders of magnitude faster in the execution time while providing solutions that are just slightly worse than the ones found by Gurobi, which are near-optimal.

We developed a mathematical model based on previous work by López, Ekin, Mediavilla, and Jimenez (2020), generalizing it and making it easier to solve in general, building an instances generator program for this model. We successfully implemented this mathematical model in a modeling language in order to be solved with an optimization solver, in this case Gurobi, which is a commercial software that we could use through the use of an academic license.

Using the p-dispersion heuristic for location and opportunity cost with the usage of efficient data structures for allocation we can offer the best performance in feasibility and speed, as well as using a local search heuristic to repair to feasibility the solutions constructed and improve these solutions. These two heuristics were adapted and integrated within the GRASP using a value-based restricted candidate

list to promote diversification. We also reduced the time complexity of the first allocation heuristic by introducing several data structures: a priority queue, a dictionary, arrays and matrices to remove repeated computations as well as partially evaluating the solution built, instead of computing everything every time.

By also using parallel computing, we reduced the times of the metaheuristic, exploiting its parallelizable nature and using modern computing infrastructure that maximizes the performance of today's processors, although for larger instances we found erratic loads in the CPU and the threads, which could be due to thermal throttling, garbage collection of the programming language or frequent cache misses which led to memory access slowdowns.

For small size instances, the heuristics sometimes struggle to find feasible solutions, for larger size instances, feasibility is achieved for all instances. However, once the heuristics are integrated in the GRASP framework, feasibility is always reached.

The results found during the computational experiments enabled us to take informed decisions while building the metaheuristic framework, allowing us to fine-tune the parameters of the GRASP algorithm to provide results with a good balance of runtime and objective function value. GRASP's solutions closely rival Gurobi's but are significantly faster.

6.2 Future work

Instead of using risk as a given parameter of the problem, we could model it as an uncertainty measure of a probability distribution that describes the probability of clients defaulting on their loans and financial services provided. By doing so, we can introduce a stochastic component to the problem, turning it into a stochastic optimization problem, in which we must generate scenarios and use estimations and samples to solve it using a robust framework, according to Spall (2003).

As stated back in Chapter 3, the problem studied in this thesis does not consider contiguity constraints. By incorporating contiguity constraints we can model the problem as a graph, more so than the current problem, and use graph theory algorithms to ensure connected territories as graphs.

We can also try to profile and run the metaheuristic controlling for certain variables to determine the cause of the extreme spikes and valleys in CPU load and temperature when running the multi-threaded GRASP in instances of Size 3. By profiling the code, its memory allocations and setting specific parameters such as voltage of the CPU and keeping track of CPU load and temperature, we can pinpoint where is the issue, allowing us to have a clear answer and a solution to the inconsistent speeds of the GRASP implementation.

Both completely change the nature of the problem as well as requiring that the solution methods are adapted to account for these changes. Thus, they provide interesting avenues of research and future work.

The local search heuristic was improved by using a cyclical approach, laying the groundwork for using a Variable Neighborhood Search metaheuristic (Hansen et al., 2010), which has been successfully implemented for territorial design problems (Quevedo-Orozco and Ríos-Mercado, 2015).

Finally, we can explore the use of other strategies to solve the problem, such as using different metaheuristics: the mentioned VNS, Tabu Search, or Scatter Search. We could even hybridize the heuristic with mathematical programming, by partially solving the problem with the heuristics and then passing it to the solver as a warm start for its algorithms.

Appendix A

Appendix

A.1 Exact solutions tables of results

For the results of Size 1, the table is found in Chapter 5.

Table A.1: Exact solver solutions for Size 2 (Instances 1-10)

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	1800	949005	944849.51	0.44
2	1800	975979	967012.29	0.92
3	1800	988670	982048.42	0.67
4	1800	974186	971396.86	0.29
5	1800	958455	957950.01	0.05
6	1800	964088	961976.21	0.22
7	1800	968567	963149.90	0.56
8	1800	986315	983207.46	0.32
9	951731	951731	947419.24	0.45
10	1800	999440	991050.29	0.84

Table A.2: Exact solver solutions for Size 2 (Instances 11-20)

Instance	Time (seconds)	z	Best Bound	% MIP Gap
11	1800	961532	954783.34	0.70
12	1800	942864	940956.26	0.20
13	1800	1012710	1009464.27	0.32
14	1800	987762	985021.26	0.28
15	1800	961675	957101.37	0.48
16	1800	990828	983915.69	0.70
17	1800	992812	991539.11	0.13
18	1800	980761	974474.12	0.64
19	1800	973562	970867.41	0.28
20	1800	972605	968213.68	0.45

Table A.3: Exact solver solutions for Size 3

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	7200	883693	881117.84	0.29
2	7200	893408	892105.54	0.15
3	7200	882602	882275.84	0.04
4	7200	913347	910961.69	0.26
5	7200	895581	893657.94	0.21
6	7200	896157	894518.51	0.18
7	7200	893684	892487.89	0.13
8	7200	895125	892662.59	0.28
9	7200	877510	876485.60	0.12
10	7200	892898	891652.12	0.14

A.2 Constructive heuristic tables of results

A very interesting thing to notice is that due to the just-in-time compilation of the Julia language, the first time a function is called, it is also compiled. For all sizes, the 10th instance was the first to be solved, thus all the times for Instance 10 include the compilation time, therefore all times will show up as outliers when compared to the other solutions' times. Still, the compilation time is minimal and has no major impact on results.

The ε parameter denotes a time shorter than 0.1 seconds. The location phase time is reported as 1.0 seconds because that is the minimum time reported by the optimizer in the Partial Linear Programming Relaxation approach.

A.2.1 Instances of Size 1

Table A.4: Results of constructive heuristics under Opp. Queue and PLP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	1	ε	False	4
2	1	ε	False	3
3	1	ε	False	3
4	1	ε	False	3
5	1	ε	False	3
6	1	ε	False	3
7	1	ε	False	3
8	1	ε	False	7
9	1	ε	False	4
10	1	0.27	False	3
11	1	ε	False	7
12	1	ε	False	6
13	1	ε	False	5
14	1	ε	False	6
15	1	ε	False	3
16	1	ε	False	10
17	1	ε	False	3
18	1	ε	False	6
19	1	ε	False	6
20	1	ε	False	4

Table A.5: Results of constructive heuristics under Opp. Queue and p-dP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	ε	False	6
2	ε	ε	False	8
3	ε	ε	False	7
4	ε	ε	False	7
5	ε	ε	False	5
6	ε	ε	False	3
7	ε	ε	False	3
8	ε	ε	False	6
9	ε	ε	False	3
10	0.55	0.27	False	6
11	ε	ε	False	3
12	ε	ε	False	6
13	ε	ε	False	9
14	ε	ε	False	6
15	ε	ε	False	6
16	ε	ε	False	3
17	ε	ε	False	3
18	ε	ε	False	6
19	ε	ε	False	5
20	ε	ε	False	7

Table A.6: Results of constructive heuristics under Opp. Matrix and PLP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	1	0.37	False	45
2	1	0.36	False	51
3	1	0.17	False	46
4	1	0.36	False	49
5	1	0.17	False	50
6	1	0.40	False	49
7	1	0.17	False	46
8	1	0.46	False	49
9	1	0.16	False	43
10	1	0.71	False	51
11	1	0.50	False	53
12	1	0.21	False	55
13	1	0.43	False	50
14	1	0.22	False	54
15	1	0.21	False	48
16	1	0.38	False	53
17	1	0.16	False	54
18	1	0.38	False	53
19	1	0.19	False	49
20	1	0.18	False	47

Table A.7: Results of constructive heuristics under Opp. Matrix and p-dP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	0.21	False	42
2	ε	0.12	False	41
3	ε	0.12	False	38
4	ε	0.12	False	47
5	ε	0.15	False	37
6	ε	0.13	False	49
7	ε	0.15	False	44
8	ε	0.15	False	41
9	ε	0.17	False	43
10	0.55	0.68	False	44
11	ε	0.25	False	40
12	ε	0.20	False	43
13	ε	0.24	False	46
14	ε	0.23	False	49
15	ε	0.24	False	45
16	ε	0.20	False	47
17	ε	0.20	False	44
18	ε	0.17	False	43
19	ε	0.23	False	42
20	ε	0.38	False	50

A.2.2 Instances of Size 2

Table A.8: Results of constructive heuristics under Opp. Queue and PLP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	12	ε	True	0
2	11	ε	True	0
3	15	ε	True	0
4	13	ε	True	0
5	10	ε	False	3
6	10	ε	True	0
7	11	ε	True	0
8	13	ε	True	0
9	11	ε	True	0
10	15	0.29	True	0
11	11	ε	True	0
12	12	ε	True	0
13	12	ε	True	0
14	11	ε	True	0
15	12	ε	True	0
16	11	0.05	False	3
17	11	ε	True	0
18	12	ε	True	0
19	11	ε	True	0
20	11	ε	True	0

Table A.9: Results of constructive heuristics under Opp. Queue and p-dP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	ε	True	0
2	ε	0.01	False	1
3	ε	ε	True	0
4	ε	ε	True	0
5	ε	ε	True	0
6	ε	ε	True	0
7	ε	ε	False	2
8	ε	ε	True	0
9	ε	ε	True	0
10	0.56	0.28	True	0
11	ε	ε	True	0
12	ε	ε	True	0
13	ε	ε	True	0
14	ε	ε	True	0
15	ε	ε	True	0
16	ε	ε	True	0
17	ε	0.01	True	0
18	ε	ε	True	0
19	ε	ε	True	0
20	ε	ε	True	0

Table A.10: Results of constructive heuristics under Opp. Matrix and PLP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	12	1.57	False	88
2	11	1.53	False	103
3	15	1.5	False	98
4	12	1.92	False	93
5	10	1.38	False	81
6	10	2.02	False	94
7	11	1.1	False	96
8	13	1.37	False	91
9	11	1.75	False	99
10	15	1.92	False	98
11	11	1.42	False	92
12	12	2.13	False	85
13	12	1.16	False	101
14	10	1.69	False	87
15	12	1.64	False	82
16	11	2.0	False	101
17	10	1.64	False	83
18	12	1.28	False	85
19	11	2.08	False	89
20	11	1.52	False	94

Table A.11: Results of constructive heuristics under Opp. Matrix and p-dP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	1.5	False	87
2	ε	1.79	False	78
3	ε	1.47	False	80
4	ε	1.77	False	79
5	ε	1.25	False	84
6	ε	1.56	False	80
7	ε	1.87	False	69
8	ε	1.64	False	76
9	ε	1.07	False	80
10	0.54	1.84	False	90
11	ε	1.6	False	73
12	ε	1.6	False	89
13	ε	1.73	False	86
14	ε	1.6	False	73
15	ε	1.6	False	81
16	ε	1.56	False	84
17	ε	1.72	False	90
18	ε	1.36	False	80
19	ε	1.79	False	81
20	ε	1.71	False	91

A.3 Local search heuristic tables of results

The ε parameter denotes a time shorter than 0.1 seconds, LS Cycles is the number of times the Local Search was performed in a cyclic way, first performing a move, moving to another one when the local optimum was reached, while any move presented an improvement over the incumbent best solution. Instances with a relative optimality gap of 100% are instances that could not be repaired to feasibility by the heuristic.

A.3.1 Instances of Size 1

Table A.12: Local Search results with Matrix, PLP and BF Strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.43	0.7	4	83.28	10.54
2	False	1.7	ε	0	0	100
3	True	0.67	0.84	4	132.84	11.75
4	False	1.87	ε	0	0	100
5	True	0.6	0.65	4	63.16	16.73
6	False	2.18	ε	0	0	100
7	False	1.71	ε	0	0	100
8	False	1.81	ε	0	0	100
9	True	0.65	0.72	5	55.01	15.14
10	True	0.63	0.75	5	59.69	7.55
11	False	1.81	ε	0	0	100
12	False	2.18	ε	0	0	100
13	True	0.65	0.56	5	51.65	10.14
14	True	0.67	0.56	4	42.0	15.37
15	True	0.88	0.69	4	90.31	7.66
16	True	0.62	0.67	5	64.56	6.36
17	True	1.02	0.72	5	80.69	12.0
18	True	1.22	1.09	7	151.49	10.09
19	True	1.08	0.72	5	77.63	8.1
20	False	1.46	ε	0	0	100

Table A.13: Local Search results with Queue, PLP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.42	0.94	6	97.14	10.35
2	True	0.14	1.14	6	155.57	9.84
3	True	0.13	1.27	5	137.57	12.14
4	True	0.1	0.86	4	88.66	15.51
5	True	0.05	1.07	4	86.66	11.14
6	True	0.1	0.81	5	79.8	8.76
7	True	0.14	1.49	4	104.18	10.9
8	False	0.37	ε	0	0	100
9	True	0.35	1.09	5	92.59	14.14
10	False	0.5	ε	0	0	100
11	False	0.83	ε	0	0	100
12	False	0.52	ε	0	0	100
13	True	0.36	0.68	5	71.65	9.05
14	True	0.17	0.77	5	53.07	16.4
15	True	0.24	1.1	7	108.33	10.2
16	False	0.79	ε	0	0	100
17	True	0.09	0.94	5	86.54	12.54
18	False	0.8	ε	0	0	100
19	True	0.08	0.74	4	65.49	12.54
20	True	0.15	0.77	4	91.43	12.3

Table A.14: Local Search results with Matrix, p-dP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.75	1.08	4	94.72	11.93
2	False	2.1	ε	0	0	100
3	True	0.32	0.57	3	47.01	11.82
4	True	0.22	0.58	5	28.64	9.93
5	True	2.05	0.6	5	36.76	10.21
6	True	0.4	1.0	5	81.2	6.34
7	True	0.32	0.64	4	34.45	11.25
8	False	1.57	ε	0	0	100
9	True	0.35	0.77	5	35.12	8.09
10	False	2.13	ε	0	0	100
11	False	1.48	ε	0	0	100
12	True	0.46	0.71	5	38.0	11.95
13	True	0.65	1.01	7	56.05	8.61
14	True	0.53	1.12	5	57.47	12.08
15	True	0.48	1.06	7	74.81	7.98
16	True	0.41	0.72	4	58.18	9.78
17	True	0.58	1.0	4	76.76	11.05
18	True	0.45	0.81	4	51.16	9.02
19	True	0.53	0.76	4	57.19	16.88
20	True	2.58	0.77	5	69.58	5.34

Table A.15: Local Search results with Queue, p-dP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	False	0.43	ε	0	0	100
2	False	0.47	ε	0	0	100
3	True	0.22	0.96	4	100.07	10.06
4	True	0.17	0.76	4	65.4	11.95
5	True	0.23	0.99	4	87.99	6.89
6	True	0.14	1.29	5	108.61	7.48
7	True	0.07	0.88	4	75.35	8.7
8	True	0.23	0.8	4	65.71	8.59
9	True	0.13	1.22	4	94.89	8.65
10	False	0.71	ε	0	0	100
11	False	0.45	ε	0	0	100
12	True	0.18	1.02	6	69.52	10.82
13	False	0.54	ε	0	0	100
14	True	0.2	1.01	3	82.39	14.51
15	True	0.24	1.28	5	95.25	11.87
16	False	0.43	ε	0	0	100
17	True	0.12	1.09	5	101.59	13.55
18	True	0.23	1.11	4	83.32	9.87
19	True	0.07	1.1	4	105.36	16.18
20	False	0.5	ε	0	0	100

Table A.16: Local Search results with Matrix, PLP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.47	0.58	6	86.21	9.11
2	False	1.72	ε	0	0	100
3	True	0.65	0.65	6	136.82	10.24
4	False	1.91	ε	0	0	100
5	True	0.56	0.57	5	71.45	12.49
6	False	2.22	ε	0	0	100
7	False	1.68	ε	0	0	100
8	False	1.86	ε	0	0	100
9	True	0.62	0.4	4	51.49	17.07
10	True	0.64	0.53	5	58.32	8.34
11	False	1.82	ε	0	0	100
12	False	2.18	ε	0	0	100
13	True	0.63	0.4	5	50.6	10.76
14	True	0.65	0.39	4	47.14	12.31
15	True	0.88	0.61	6	83.38	11.02
16	True	0.68	0.42	4	64.29	6.52
17	True	0.97	0.4	4	80.89	11.9
18	True	1.28	0.59	5	154.55	8.99
19	True	1.04	0.51	5	72.94	10.53
20	False	1.45	ε	0	0	100

Table A.17: Local Search results with Queue, PLP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.2	0.41	4	69.72	22.82
2	True	0.11	0.28	3	75.38	38.13
3	True	0.09	0.34	3	79.91	33.47
4	True	0.08	0.64	5	35.05	39.52
5	True	0.07	0.34	3	60.43	23.62
6	True	0.08	0.56	3	47.37	25.22
7	True	0.15	0.33	3	73.59	24.25
8	False	0.59	ε	0	0	100
9	True	0.33	0.25	4	53.07	31.76
10	False	0.48	ε	0	0	100
11	False	0.63	ε	0	0	100
12	False	0.51	ε	0	0	100
13	True	0.12	0.51	3	37.38	27.21
14	True	0.17	0.6	7	35.98	25.74
15	True	0.24	0.42	2	27.63	44.99
16	False	0.51	ε	0	0	100
17	True	0.11	0.34	3	50.89	29.25
18	False	0.59	ε	0	0	100
19	True	0.06	0.39	4	48.21	21.67
20	True	0.14	0.4	2	21.86	44.17

Table A.18: Local Search results with Matrix, p-dP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.75	0.59	4	96.41	11.16
2	False	2.11	ε	0	0	100
3	True	0.32	0.33	4	41.63	15.05
4	True	0.23	0.43	5	27.22	10.92
5	True	2.05	0.5	4	40.1	8.02
6	True	0.4	0.69	5	80.41	6.75
7	True	0.32	0.39	4	36.99	9.57
8	False	1.58	ε	0	0	100
9	True	0.35	0.7	6	38.8	5.58
10	False	2.13	ε	0	0	100
11	False	1.47	ε	0	0	100
12	True	0.46	0.59	6	36.82	12.7
13	True	0.66	0.52	4	45.92	14.54
14	True	0.53	0.67	5	58.28	11.63
15	True	0.47	0.73	6	76.06	7.33
16	True	0.39	0.5	4	62.88	7.1
17	True	0.58	0.64	6	81.71	8.56
18	True	0.45	0.62	5	53.05	7.88
19	True	0.52	0.49	4	58.16	16.37
20	True	2.58	0.56	5	68.62	5.88

Table A.19: Local Search results with Queue, p-dP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	False	0.43	ε	0	0	100
2	False	0.48	ε	0	0	100
3	True	0.22	0.55	4	96.7	11.58
4	True	0.18	0.52	4	68.19	10.47
5	True	0.23	0.56	5	84.28	8.73
6	True	0.14	0.66	4	113.09	5.49
7	True	0.08	0.49	3	75.21	8.77
8	True	0.22	0.53	5	64.51	9.25
9	True	0.13	0.54	4	96.25	8.02
10	False	0.72	ε	0	0	100
11	False	0.43	ε	0	0	100
12	True	0.18	0.64	5	74.78	8.05
13	False	0.54	ε	0	0	100
14	True	0.2	0.56	4	80.09	15.59
15	True	0.24	0.78	6	95.25	11.87
16	False	0.43	ε	0	0	100
17	True	0.1	0.88	6	109.43	10.19
18	True	0.21	0.57	5	89.44	6.86
19	True	0.09	0.74	5	103.97	16.75
20	False	0.51	ε	0	0	100

A.3.2 Instances of Size 2

Table A.20: Local Search results with Matrix, PLP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	5.73	7.92	5	77.64	13.51
2	True	10.97	10.47	5	114.95	11.36
3	True	8.86	8.09	5	71.05	14.36
4	True	5.35	8.73	5	65.89	13.11
5	True	9.84	7.51	4	72.25	9.8
6	True	7.57	9.99	5	88.57	10.78
7	True	4.74	6.67	5	53.97	12.44
8	True	4.99	6.72	6	42.42	10.86
9	True	9.38	8.44	4	91.65	11.02
10	True	10.77	9.5	5	81.07	11.99
11	True	8.79	8.58	6	98.37	8.43
12	True	8.12	6.65	4	69.29	14.16
13	True	7.5	13.07	5	116.47	13.08
14	True	4.46	7.2	5	65.05	10.73
15	True	3.03	5.16	4	18.67	9.75
16	True	6.45	7.25	4	67.63	10.06
17	True	4.27	6.94	5	38.03	8.25
18	True	5.2	7.51	5	57.29	11.23
19	True	5.19	8.57	5	64.51	10.23
20	True	5.7	8.14	4	79.64	8.6

Table A.21: Local Search results with Matrix, p-dP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	2.13	6.26	4	27.3	8.56
2	True	4.35	10.01	5	58.88	8.02
3	True	8.15	13.28	5	68.24	11.25
4	True	3.47	9.37	5	46.52	9.94
5	True	2.82	7.2	4	31.95	10.7
6	True	3.29	7.87	5	40.04	8.44
7	True	3.72	9.33	4	46.89	9.15
8	True	3.35	8.64	5	39.06	13.36
9	True	5.0	10.57	4	56.92	12.4
10	True	4.88	9.28	6	36.41	14.37
11	True	4.09	9.48	5	51.21	8.88
12	True	6.1	9.45	4	59.81	11.16
13	True	3.83	12.53	5	49.78	13.03
14	True	3.07	9.52	5	54.22	7.8
15	True	2.63	9.01	6	52.13	8.72
16	True	5.99	11.46	6	60.72	9.96
17	True	2.69	10.07	5	46.67	9.87
18	True	3.14	8.76	5	45.84	10.74
19	True	3.08	6.81	4	37.12	10.69
20	True	7.81	11.72	4	73.31	10.68

Table A.22: Local Search results with Queue, PLP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	10.3	5	140.11	9.79
2	True	ε	13.14	5	151.9	12.47
3	True	ε	9.91	4	130.8	8.99
4	True	ε	10.47	5	110.77	15.82
5	True	0.03	9.2	4	107.28	8.89
6	True	ε	10.75	6	134.83	11.82
7	True	ε	8.59	4	99.17	10.91
8	True	ε	8.64	6	78.09	10.7
9	True	ε	12.81	5	148.45	12.73
10	True	ε	12.03	5	118.41	13.04
11	True	ε	11.94	5	153.94	6.2
12	True	ε	8.19	4	92.86	14.6
13	True	ε	14.93	5	195.0	8.39
14	True	ε	10.67	5	105.73	10.92
15	True	ε	9.55	5	68.18	9.93
16	True	0.15	10.81	5	114.79	7.97
17	True	ε	11.34	5	95.16	8.38
18	True	ε	8.57	5	97.3	11.51
19	True	ε	13.04	5	108.97	12.26
20	True	ε	10.13	5	116.31	9.08

Table A.23: Local Search results with Queue, p-dP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	12.59	4	104.29	8.87
2	True	0.1	15.86	5	119.76	10.18
3	True	ε	17.05	5	110.68	10.68
4	True	ε	17.2	6	122.46	8.55
5	True	ε	12.68	4	93.14	13.91
6	True	ε	14.55	5	116.24	9.14
7	True	0.02	16.05	4	115.43	9.6
8	True	ε	14.09	4	108.17	9.81
9	True	ε	16.26	5	119.09	12.1
10	True	ε	16.22	5	128.94	11.44
11	True	ε	14.26	4	123.93	8.47
12	True	ε	14.41	5	136.5	11.63
13	True	ε	19.25	5	137.3	12.16
14	True	ε	16.48	5	123.47	11.84
15	True	ε	15.17	5	147.18	9.98
16	True	ε	16.81	5	133.85	11.07
17	True	ε	15.81	5	122.46	8.68
18	True	ε	13.5	4	114.57	11.35
19	True	ε	15.51	5	124.4	11.24
20	True	ε	15.79	5	121.35	11.86

Table A.24: Local Search results with Matrix, PLP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	5.73	4.45	4	75.6	14.5
2	True	11.57	5.72	5	114.32	11.62
3	True	8.93	5.62	4	70.51	14.63
4	True	5.64	5.78	4	66.91	12.58
5	True	10.04	5.14	5	66.59	12.77
6	True	7.54	5.97	5	84.18	12.86
7	True	4.65	4.96	5	55.78	11.41
8	True	5.01	4.77	5	43.86	9.96
9	True	8.81	5.17	5	89.65	11.95
10	True	10.63	6.05	5	76.24	14.33
11	True	8.01	5.29	5	98.34	8.44
12	True	8.9	4.57	5	68.2	14.71
13	True	7.04	7.71	5	126.24	9.15
14	True	4.82	4.6	4	61.63	12.58
15	True	3.07	3.75	4	20.84	8.1
16	True	6.83	6.43	6	67.83	9.95
17	True	4.28	4.86	5	40.83	6.38
18	True	5.12	4.74	4	54.98	12.53
19	True	5.53	5.82	6	61.24	12.01
20	True	5.43	5.84	5	78.65	9.11

Table A.25: Local Search results with Matrix, p-dP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	2.15	4.3	5	26.75	8.95
2	True	4.35	6.15	5	53.53	11.11
3	True	8.03	7.6	5	63.85	13.57
4	True	3.48	5.59	6	45.72	10.43
5	True	2.81	4.28	5	28.34	13.14
6	True	3.27	4.2	4	36.84	10.53
7	True	3.77	6.29	6	48.22	8.33
8	True	3.37	4.68	5	40.54	12.44
9	True	5.02	6.49	6	65.82	7.43
10	True	4.83	5.25	5	37.42	13.73
11	True	4.08	6.15	6	55.38	6.38
12	True	6.07	5.0	4	57.21	12.61
13	True	3.85	6.77	5	50.4	12.67
14	True	3.06	5.27	5	53.78	8.07
15	True	2.62	5.96	7	53.34	8.0
16	True	6.03	7.67	6	60.25	10.22
17	True	2.7	5.57	5	46.39	10.04
18	True	3.15	5.44	5	46.36	10.42
19	True	3.04	4.33	4	35.84	11.52
20	True	7.82	7.59	6	72.14	11.28

Table A.26: Local Search results with Queue, PLP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	5.92	5	141.92	9.11
2	True	ε	6.78	7	152.93	12.12
3	True	ε	7.2	6	118.29	13.92
4	True	ε	7.4	5	121.3	11.61
5	True	0.03	5.32	4	109.84	7.77
6	True	ε	7.49	5	133.86	12.18
7	True	ε	5.34	6	100.44	10.34
8	True	ε	4.71	5	79.94	9.78
9	True	ε	6.61	6	160.4	8.53
10	True	ε	6.41	5	118.95	12.82
11	True	ε	7.21	5	150.7	7.4
12	True	ε	4.95	4	101.45	10.8
13	True	ε	8.48	4	188.72	10.33
14	True	ε	5.56	4	101.75	12.64
15	True	ε	5.93	5	70.52	8.68
16	True	0.3	6.8	5	110.28	9.91
17	True	ε	5.89	5	91.29	10.2
18	True	ε	6.19	6	105.77	7.71
19	True	ε	7.5	5	117.24	8.79
20	True	ε	6.75	6	114.88	9.68

Table A.27: Local Search results with Queue, p-dP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	6.89	4	104.38	8.83
2	True	0.1	7.8	4	122.31	9.14
3	True	ε	9.05	4	104.6	13.26
4	True	ε	8.96	6	120.51	9.35
5	True	ε	6.59	5	96.45	12.44
6	True	ε	7.66	5	118.16	8.33
7	True	0.02	7.9	5	113.77	10.3
8	True	ε	7.9	5	106.91	10.36
9	True	ε	6.92	5	125.39	9.57
10	True	ε	7.83	7	130.59	10.8
11	True	ε	7.16	4	123.11	8.81
12	True	ε	7.43	6	139.79	10.4
13	True	ε	9.28	5	140.84	10.85
14	True	ε	8.39	5	129.4	9.51
15	True	ε	8.14	5	148.46	9.51
16	True	ε	8.37	5	139.87	8.78
17	True	ε	7.61	6	124.48	7.85
18	True	ε	7.26	5	113.92	11.62
19	True	ε	7.78	5	125.24	10.9
20	True	ε	10.02	5	128.47	9.02

A.4 GRASP parameter tuning tables

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.1	0.1	25	6	7.896	11.21	3.35	4.696	11.45
0.1	0.1	50	5.43	6.829	8.71	7.04	7.518	7.93
0.1	0.1	75	5.75	7.024	9.31	9.94	11.239	12.25
0.1	0.1	100	4.79	6.03	8.22	14.16	14.88	15.72
0.1	0.3	25	4.42	6.815	8.37	3.34	3.969	4.73
0.1	0.3	50	5.01	6.03	7.49	6.55	7.288	8.04
0.1	0.3	75	5.12	5.78	6.96	10.27	11.129	11.95
0.1	0.3	100	5.16	6.147	7.27	12.82	14.183	15.71
0.1	0.5	25	4.2	6.096	8.32	3.51	3.947	4.28
0.1	0.5	50	5.54	6.375	7.19	6.36	7.305	8.04
0.1	0.5	75	5.21	5.747	7.1	10.49	11.306	12.05
0.1	0.5	100	4.7	5.771	7.52	12.87	14.155	15.83
0.1	0.7	25	5.28	6.468	8.47	3.58	4.072	4.4
0.1	0.7	50	4.69	5.985	6.62	6.76	7.582	8.23
0.1	0.7	75	4.63	5.889	7.41	10.78	11.374	12.2
0.1	0.7	100	4.47	5.956	7.81	13.65	14.947	15.96
0.3	0.1	25	5.97	8.066	10.75	3.79	4.019	4.47
0.3	0.1	50	5.31	7.519	10.19	7.12	7.689	8.14
0.3	0.1	75	5.87	7.121	8.55	10.66	10.998	11.28
0.3	0.1	100	5.5	6.791	8.78	14.04	14.512	14.99
0.3	0.3	25	5.61	7.668	8.9	3.37	3.956	4.47
0.3	0.3	50	5.84	7.037	9.19	7.09	7.695	8.48

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.3	0.3	75	4.73	6.515	7.86	10.54	11.158	12.28
0.3	0.3	100	4.52	6.055	6.76	13.73	14.758	16.31
0.3	0.5	25	5.69	7.018	7.72	3.75	4.158	4.5
0.3	0.5	50	5.97	6.936	8.81	7.37	7.73	8.12
0.3	0.5	75	5.62	6.843	7.63	10.11	11.197	12.28
0.3	0.5	100	5.54	6.586	7.7	13.03	14.686	15.7
0.3	0.7	25	5.1	6.985	8.68	3.7	4.089	4.51
0.3	0.7	50	5.31	6.603	8.15	7.26	7.825	8.49
0.3	0.7	75	4.63	6.255	7.68	10.78	11.531	12.74
0.3	0.7	100	4.48	6.11	7.42	14.04	14.573	15.28
0.5	0.1	25	5.59	8.195	9.55	3.73	4.054	4.36
0.5	0.1	50	4.79	7.193	8.55	6.74	7.606	8.36
0.5	0.1	75	5.36	7.154	8.57	10.22	11.051	11.75
0.5	0.1	100	4.57	7.051	9.15	13.73	14.573	15.2
0.5	0.3	25	5.64	7.158	9.1	3.3	3.996	4.53
0.5	0.3	50	5.69	6.89	8.16	7	7.57	8.28
0.5	0.3	75	4.67	6.726	8.44	10.14	10.903	12.39
0.5	0.3	100	4.25	6.099	7.24	13.22	14.566	18.5
0.5	0.5	25	5.28	7.282	9.04	3.7	4.13	4.57
0.5	0.5	50	4.65	6.99	9	6.53	7.522	8.02
0.5	0.5	75	5.21	6.745	7.62	10.31	11.273	12.66
0.5	0.5	100	5.7	6.464	8.08	14	14.592	15.28
0.5	0.7	25	5.75	7.308	9.04	3.62	4.096	4.53
0.5	0.7	50	4.16	5.862	8.19	7.12	7.775	8.45
0.5	0.7	75	5.33	6.564	7.82	9.96	11.178	12.19

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.5	0.7	100	4.81	6.34	8.44	13.57	14.87	15.71
0.7	0.1	25	5.05	7.44	10.09	3.68	4.072	4.56
0.7	0.1	50	5.29	7.455	9.23	6.76	7.598	8.06
0.7	0.1	75	5.63	7.264	9.04	10.02	11.089	12.33
0.7	0.1	100	5.17	6.342	7.76	13.35	14.3	15.9
0.7	0.3	25	5.82	7.294	9.92	3.49	3.93	4.37
0.7	0.3	50	4.84	6.401	8.29	6.74	7.573	8.19
0.7	0.3	75	5.16	6.595	7.79	9.8	11.004	11.89
0.7	0.3	100	5.51	6.382	7.21	13.77	14.398	15.89
0.7	0.5	25	5.97	7.397	9.19	3.85	4.32	5.28
0.7	0.5	50	5.23	6.451	8.54	6.96	7.39	7.92
0.7	0.5	75	5.27	6.446	8.59	10.4	11.25	12.21
0.7	0.5	100	5.46	6.69	8.12	13.65	14.416	15.63
0.7	0.7	25	5.41	7.697	10.78	3.44	4.185	4.72
0.7	0.7	50	4.75	6.782	8.54	7.1	7.699	8.19
0.7	0.7	75	5.2	6.49	7.82	9.92	11.224	12.19
0.7	0.7	100	5.11	6.081	7.31	13.6	14.95	16.36

Table A.28: GRASP fine-tuning results for Size 1 instances

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.1	0.1	25	6.17	7.53	8.76	37.26	41.36	46.98
0.1	0.1	50	6.6	7.67	9.77	63.95	71.56	79.46
0.1	0.1	75	6.27	7.55	9.44	96.33	106.04	118.2
0.1	0.1	100	5.47	7.24	9.05	131.78	142.35	157.37
0.1	0.3	25	7.19	8.33	9.15	31.48	36.86	40.45
0.1	0.3	50	6.45	7.74	8.78	64.92	72.85	83.03
0.1	0.3	75	6.43	7.33	7.99	94.64	106.66	124.66
0.1	0.3	100	6.39	7.34	8.5	120.39	140.84	157.77
0.1	0.5	25	6.6	8.12	9.46	32.81	38.5	43.95
0.1	0.5	50	6.22	7.43	8.43	65.44	75.54	86.64
0.1	0.5	75	6.8	7.86	8.98	101.89	113.15	127.44
0.1	0.5	100	6.41	7.5	8.95	130.38	147.69	165.84
0.1	0.7	25	6.69	7.95	9.26	36.96	41.03	45.56
0.1	0.7	50	6.72	7.86	8.65	72.18	80.47	91.86
0.1	0.7	75	7.18	7.74	8.19	107.41	119.37	138.98
0.1	0.7	100	6.42	7.36	8.42	142.22	156.53	175.53
0.3	0.1	25	7.86	8.79	10.93	33.02	37.56	41.24
0.3	0.1	50	6.82	8.15	9.49	67.69	74.65	80.05
0.3	0.1	75	5.97	7.73	10.15	100.14	110.55	120.57
0.3	0.1	100	6.59	7.26	8.44	132.24	143.76	154.68
0.3	0.3	25	6.82	7.91	10.39	33.83	38.13	42.63
0.3	0.3	50	6.36	8.11	9.91	69.59	74.87	81.13
0.3	0.3	75	6.18	7.66	8.46	97.61	111.18	125.26
0.3	0.3	100	6.66	7.51	8.91	129.13	147.61	165.13

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.3	0.5	25	7.74	8.32	9.11	35.48	40.51	46.51
0.3	0.5	50	7.45	8.51	9.81	70.67	79.19	90.1
0.3	0.5	75	7.18	8.26	9.21	104.26	116.91	128.22
0.3	0.5	100	6.61	7.97	8.69	136.05	152.9	172.34
0.3	0.7	25	8.04	8.74	9.47	38.87	42.52	47.25
0.3	0.7	50	7.13	8.68	10.06	76.55	82.98	92.63
0.3	0.7	75	6.58	7.69	8.85	111.01	124.56	139.04
0.3	0.7	100	6.17	7.4	8.7	150.06	164.71	187.28
0.5	0.1	25	7.35	8.69	10.69	33.52	38.08	40.38
0.5	0.1	50	6.67	7.82	9.97	68.5	75.69	81.67
0.5	0.1	75	6.59	7.75	9.22	103.33	113.05	118.93
0.5	0.1	100	5.74	7.58	9.86	135.77	147.85	155.75
0.5	0.3	25	6.77	8.3	9.29	36.01	38.93	44.05
0.5	0.3	50	6.24	8.15	9.42	70.85	76.95	82.05
0.5	0.3	75	6.7	7.94	9.33	105.21	113.98	125.98
0.5	0.3	100	6.79	7.72	8.69	139.64	149.68	165.18
0.5	0.5	25	7.57	8.73	9.76	36.87	40.75	45.75
0.5	0.5	50	6.75	8.14	9.43	74.11	80.22	87.49
0.5	0.5	75	7.37	7.94	9.4	110.98	119.89	131.17
0.5	0.5	100	5.87	7.8	8.69	145.64	157.69	178.4
0.5	0.7	25	7.4	8.65	9.85	37.52	43.5	49.58
0.5	0.7	50	4.88	7.66	8.87	78.07	86.77	97.81
0.5	0.7	75	6.6	8.02	9.21	117.06	127.2	144.39
0.5	0.7	100	6.53	7.71	8.39	151.83	167.12	184.81
0.7	0.1	25	7.28	8.55	11.08	36.07	39.18	41.55

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.7	0.1	50	6.49	7.64	10.35	70.86	77.73	81.21
0.7	0.1	75	6.53	7.49	9.26	106.47	114.23	123.11
0.7	0.1	100	5.43	7.08	9.54	136.44	151.39	158.22
0.7	0.3	25	7.16	8.59	9.47	36.87	39.57	44.23
0.7	0.3	50	7.4	7.91	8.8	74.33	79.89	89.21
0.7	0.3	75	6.28	7.53	9.49	106.64	117.07	127.05
0.7	0.3	100	5.69	7.73	9.64	140.18	153.92	169.95
0.7	0.5	25	6.68	8.55	11.24	39.56	41.34	44.64
0.7	0.5	50	6.66	7.85	9.02	74.17	80.61	88.41
0.7	0.5	75	6.85	7.77	8.52	110.73	121.06	130.91
0.7	0.5	100	5.85	7.62	8.84	142.74	160.38	179.97
0.7	0.7	25	7.81	8.8	10.17	41.24	44.47	48.08
0.7	0.7	50	7.53	8.5	9.49	78.36	87.07	94.29
0.7	0.7	75	6.39	7.44	8.33	120.84	130.08	143.76
0.7	0.7	100	4.49	7.49	8.99	156.56	170.61	187.39

Table A.29: GRASP fine-tuning results for Size 2 instances

A.5 GRASP Statistical tests

A.5.1 Statistical testing of Thread Tuning for Size 1 instances

Threads	Mean Runtime (s)	Standard Deviation (s)
1	38.28	1.88
2	33.47	2.11
3	24.32	1.39
4	22.96	1.25
5	19.00	0.98
6	18.62	0.91
7	17.01	0.98
8	16.89	0.66
9	16.09	0.70
10	15.60	0.62
11	15.32	0.77
12	15.34	0.68
13	14.82	0.61
14	14.74	0.42
15	14.48	0.49
16	14.60	0.64

Table A.30: Thread performance comparison for Size 1 instances

Repeated Measures ANOVA Summary

Source	F Value	Num DF	Den DF	Pr $> F$
Threads	1695.6740	15.0000	285.0000	0.0000

Table A.31: Summary of Repeated Measures ANOVA

The table contains the following:

Source: Refers to the independent variable or factor being tested, which in this case is “Threads”, F Value: Represents the test statistic calculated by the ANOVA, num DF: Indicates the degrees of freedom for the numerator, which is the number of levels of the independent variable minus 1, den DF: Represents the degrees of freedom for the denominator, which is the total number of observations minus the number of levels of the independent variable, Pr $> F$: Denotes the p-value associated with the F statistic, indicating the statistical significance of the results.

The repeated measures ANOVA tests if there are statistically significant differences in the means of the same subjects under different conditions (in this case, runtime under different thread counts). The results show that there is a statistically significant difference in runtimes across different thread counts.

Following a significant ANOVA result, the Tukey’s HSD post-hoc test helps identify which specific group(s) differ from each other. Tukey’s HSD (Honestly Significant Difference) post-hoc test is a multiple comparison test used to find which means among a set of means differ from the rest. It is commonly used after a significant ANOVA result to determine which groups are different. The test compares all possible pairs of means and controls the familywise error rate.

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
1	2	-4809.85	0.0	-5962.0556	-3657.6444	True
1	3	-13961.85	0.0	-15114.0556	-12809.6444	True
1	4	-15317.35	0.0	-16469.5556	-14165.1444	True
1	5	-19279.9	0.0	-20432.1056	-18127.6944	True
1	6	-19662.85	0.0	-20815.0556	-18510.6444	True
1	7	-21270.95	0.0	-22423.1556	-20118.7444	True
1	8	-21387.25	0.0	-22539.4556	-20235.0444	True
1	9	-22187.1	0.0	-23339.3056	-21034.8944	True
1	10	-22683.45	0.0	-23835.6556	-21531.2444	True
1	11	-22960.0	0.0	-24112.2056	-21807.7944	True
1	12	-22937.5	0.0	-24089.7056	-21785.2944	True
1	13	-23461.05	0.0	-24613.2556	-22308.8444	True
1	14	-23545.4	0.0	-24697.6056	-22393.1944	True
1	15	-23803.15	0.0	-24955.3556	-22650.9444	True
1	16	-23681.65	0.0	-24833.8556	-22529.4444	True
2	3	-9152.0	0.0	-10304.2056	-7999.7944	True
2	4	-10507.5	0.0	-11659.7056	-9355.2944	True
2	5	-14470.05	0.0	-15622.2556	-13317.8444	True
2	6	-14853.0	0.0	-16005.2056	-13700.7944	True
2	7	-16461.1	0.0	-17613.3056	-15308.8944	True
2	8	-16577.4	0.0	-17729.6056	-15425.1944	True
2	9	-17377.25	0.0	-18529.4556	-16225.0444	True
2	10	-17873.6	0.0	-19025.8056	-16721.3944	True
2	11	-18150.15	0.0	-19302.3556	-16997.9444	True
2	12	-18127.65	0.0	-19279.8556	-16975.4444	True
2	13	-18651.2	0.0	-19803.4056	-17498.9944	True
2	14	-18735.55	0.0	-19887.7556	-17583.3444	True
2	15	-18993.3	0.0	-20145.5056	-17841.0944	True
2	16	-18871.8	0.0	-20024.0056	-17719.5944	True
3	4	-1355.5	0.0061	-2507.7056	-203.2944	True
3	5	-5318.05	0.0	-6470.2556	-4165.8444	True
3	6	-5701.0	0.0	-6853.2056	-4548.7944	True
3	7	-7309.1	0.0	-8461.3056	-6156.8944	True
3	8	-7425.4	0.0	-8577.6056	-6273.1944	True
3	9	-8225.25	0.0	-9377.4556	-7073.0444	True
3	10	-8721.6	0.0	-9873.8056	-7569.3944	True
3	11	-8998.15	0.0	-10150.3556	-7845.9444	True
3	12	-8975.65	0.0	-10127.8556	-7823.4444	True
3	13	-9499.2	0.0	-10651.4056	-8346.9944	True
3	14	-9583.55	0.0	-10735.7556	-8431.3444	True
3	15	-9841.3	0.0	-10993.5056	-8689.0944	True
3	16	-9719.8	0.0	-10872.0056	-8567.5944	True

Table A.32: Tukey HSD test results for threads Size 1 Part 1

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
4	5	-3962.55	0.0	-5114.7556	-2810.3444	True
4	6	-4345.5	0.0	-5497.7056	-3193.2944	True
4	7	-5953.6	0.0	-7105.8056	-4801.3944	True
4	8	-6069.9	0.0	-7222.1056	-4917.6944	True
4	9	-6869.75	0.0	-8021.9556	-5717.5444	True
4	10	-7366.1	0.0	-8518.3056	-6213.8944	True
4	11	-7642.65	0.0	-8794.8556	-6490.4444	True
4	12	-7620.15	0.0	-8772.3556	-6467.9444	True
4	13	-8143.7	0.0	-9295.9056	-6991.4944	True
4	14	-8228.05	0.0	-9380.2556	-7075.8444	True
4	15	-8485.8	0.0	-9638.0056	-7333.5944	True
4	16	-8364.3	0.0	-9516.5056	-7212.0944	True
5	6	-382.95	0.9989	-1535.1556	769.2556	False
5	7	-1991.05	0.0	-3143.2556	-838.8444	True
5	8	-2107.35	0.0	-3259.5556	-955.1444	True
5	9	-2907.2	0.0	-4059.4056	-1754.9944	True
5	10	-3403.55	0.0	-4555.7556	-2251.3444	True
5	11	-3680.1	0.0	-4832.3056	-2527.8944	True
5	12	-3657.6	0.0	-4809.8056	-2505.3944	True
5	13	-4181.15	0.0	-5333.3556	-3028.9444	True
5	14	-4265.5	0.0	-5417.7056	-3113.2944	True
5	15	-4523.25	0.0	-5675.4556	-3371.0444	True
5	16	-4401.75	0.0	-5553.9556	-3249.5444	True
6	7	-1608.1	0.0003	-2760.3056	-455.8944	True
6	8	-1724.4	0.0	-2876.6056	-572.1944	True
6	9	-2524.25	0.0	-3676.4556	-1372.0444	True
6	10	-3020.6	0.0	-4172.8056	-1868.3944	True
6	11	-3297.15	0.0	-4449.3556	-2144.9444	True
6	12	-3274.65	0.0	-4426.8556	-2122.4444	True
6	13	-3798.2	0.0	-4950.4056	-2645.9944	True
6	14	-3882.55	0.0	-5034.7556	-2730.3444	True
6	15	-4140.3	0.0	-5292.5056	-2988.0944	True
6	16	-4018.8	0.0	-5171.0056	-2866.5944	True
7	8	-116.3	1.0	-1268.5056	1035.9056	False
7	9	-916.15	0.306	-2068.3556	236.0556	False
7	10	-1412.5	0.0031	-2564.7056	-260.2944	True
7	11	-1689.05	0.0001	-2841.2556	-536.8444	True
7	12	-1666.55	0.0001	-2818.7556	-514.3444	True
7	13	-2190.1	0.0	-3342.3056	-1037.8944	True
7	14	-2274.45	0.0	-3426.6556	-1122.2444	True
7	15	-2532.2	0.0	-3684.4056	-1379.9944	True
7	16	-2410.7	0.0	-3562.9056	-1258.4944	True

Table A.33: Tukey HSD test results for threads Size 1 Part 2

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
8	9	-799.85	0.5502	-1952.0556	352.3556	False
8	10	-1296.2	0.0117	-2448.4056	-143.9944	True
8	11	-1572.75	0.0004	-2724.9556	-420.5444	True
8	12	-1550.25	0.0005	-2702.4556	-398.0444	True
8	13	-2073.8	0.0	-3226.0056	-921.5944	True
8	14	-2158.15	0.0	-3310.3556	-1005.9444	True
8	15	-2415.9	0.0	-3568.1056	-1263.6944	True
8	16	-2294.4	0.0	-3446.6056	-1142.1944	True
9	10	-496.35	0.9832	-1648.5556	655.8556	False
9	11	-772.9	0.6109	-1925.1056	379.3056	False
9	12	-750.4	0.6606	-1902.6056	401.8056	False
9	13	-1273.95	0.0149	-2426.1556	-121.7444	True
9	14	-1358.3	0.0059	-2510.5056	-206.0944	True
9	15	-1616.05	0.0002	-2768.2556	-463.8444	True
9	16	-1494.55	0.0011	-2646.7556	-342.3444	True
10	11	-276.55	1.0	-1428.7556	875.6556	False
10	12	-254.05	1.0	-1406.2556	898.1556	False
10	13	-777.6	0.6003	-1929.8056	374.6056	False
10	14	-861.95	0.4133	-2014.1556	290.2556	False
10	15	-1119.7	0.0671	-2271.9056	32.5056	False
10	16	-998.2	0.1781	-2150.4056	154.0056	False
11	12	22.5	1.0	-1129.7056	1174.7056	False
11	13	-501.05	0.9816	-1653.2556	651.1556	False
11	14	-585.4	0.9297	-1737.6056	566.8056	False
11	15	-843.15	0.4537	-1995.3556	309.0556	False
11	16	-721.65	0.7214	-1873.8556	430.5556	False
12	13	-523.55	0.9725	-1675.7556	628.6556	False
12	14	-607.9	0.9064	-1760.1056	544.3056	False
12	15	-865.65	0.4055	-2017.8556	286.5556	False
12	16	-744.15	0.6741	-1896.3556	408.0556	False
13	14	-84.35	1.0	-1236.5556	1067.8556	False
13	15	-342.1	0.9997	-1494.3056	810.1056	False
13	16	-220.6	1.0	-1372.8056	931.6056	False
14	15	-257.75	1.0	-1409.9556	894.4556	False
14	16	-136.25	1.0	-1288.4556	1015.9556	False
15	16	121.5	1.0	-1030.7056	1273.7056	False

Table A.34: Tukey HSD test results for threads Size 1 Part 3

The results of the Tukey Honestly Significant Difference (HSD) test reveal significant differences in the mean values of the dependent variable across different groups (Groups 1 to 16). Specifically:

There is a statistically significant difference ($p < 0.05$) in the mean values of the dependent variable between Group 1 and all other groups (Groups 2 to 16). Significant differences exist among most pairs of groups, with a few exceptions: a) There is no statistically significant difference between Group 5 and Group 6 ($p = 0.9989$). b) Group 7 does not significantly differ from Group 8 ($p = 1.0$) or Group 9 ($p = 0.306$). c) Group 8 does not significantly differ from Group 9 ($p = 0.5502$). d) Groups 9, 10, 11, and 12 do not significantly differ from each other (all $p > 0.05$). e) Groups 10 through 16 show no significant differences among themselves (all $p > 0.05$). The largest mean differences are observed between Group 1 and the higher-numbered groups, with the difference increasing as the group number increases. From Group 10 onwards, the differences between consecutive groups become smaller and statistically insignificant.

These findings suggest that the number of threads significantly influences the mean values of the dependent variable, which in this case it is the runtime, with Group 1 showing significant differences compared to all other groups, while certain pairs of groups exhibit no discernible differences. These results provide valuable insights into the relationship between the number of threads and the dependent variable, contributing to a deeper understanding of the experimental conditions under investigation.

Even though on average, the runtime does go down as more threads are added, the improvement eventually plateaus.

Effect Sizes (Cohen's d compared to 1 thread): Effect size is a measure of the magnitude of the difference between groups. Here, Cohen's d is calculated to quantify the difference in runtimes between 1 thread and the other thread counts.

Comparison	**Cohen's d**
Group 1 vs. Group 2	2.40
Group 1 vs. Group 3	8.44
Group 1 vs. Group 4	9.58
Group 1 vs. Group 5	12.82
Group 1 vs. Group 6	13.29
Group 1 vs. Group 7	14.15
Group 1 vs. Group 8	15.16
Group 1 vs. Group 9	15.62
Group 1 vs. Group 10	16.17
Group 1 vs. Group 11	15.95
Group 1 vs. Group 12	16.19
Group 1 vs. Group 13	16.75
Group 1 vs. Group 14	17.25
Group 1 vs. Group 15	17.28
Group 1 vs. Group 16	16.82

Table A.35: Cohen's d effect sizes for pairwise comparisons

These effect sizes indicate large differences between the mean values of the dependent variable for Group 1 compared to each of the other groups. The magnitude of these differences, as measured by Cohen's d, is substantial, suggesting a strong effect of the number of threads on the dependent variable.

Paired t-tests between adjacent thread counts: The paired t-tests are used to compare the means of the same group at two different times (in this case, runtimes between adjacent thread counts). The results provide insights into the significance of the performance differences between consecutive thread counts.

Pair	T**-Value**	p**-value**
1 vs. 2	15.07	5.07×10^{-12}
2 vs. 3	26.32	2.06×10^{-16}
3 vs. 4	6.29	4.89×10^{-6}
4 vs. 5	18.68	1.09×10^{-13}
5 vs. 6	1.99	0.0616
6 vs. 7	10.32	3.17×10^{-9}
7 vs. 8	0.49	0.6311
8 vs. 9	4.46	2.66×10^{-4}
9 vs. 10	4.01	7.48×10^{-4}
10 vs. 11	2.21	0.0399
11 vs. 12	-0.16	0.8779
12 vs. 13	3.68	0.0016
13 vs. 14	0.70	0.4947
14 vs. 15	2.68	0.0147
15 vs. 16	-0.78	0.4439

Table A.36: Paired t-tests results

Our paired t-tests indicate significant differences in mean values of the dependent variable between several pairs of groups. Specifically:

The mean values of Group 1 and Group 2 differ significantly ($t = 15.07, p < 0.001$). Significant differences are also observed between Group 2 and Group 3 ($t = 26.32, p < 0.001$), Group 3 and Group 4 ($t = 6.29, p < 0.001$), and Group 4 and Group 5 ($t = 18.68, p < 0.001$). The difference between Group 5 and Group 6 is not statistically significant at the conventional 0.05 level, but it is close ($t = 1.99, p = 0.0616$). Significant differences re-emerge between Group 6 and Group 7 ($t = 10.32, p < 0.001$). No significant difference is observed between Group 7 and

Group 8 ($t = 0.49, p = 0.6311$). Significant differences are found again between Group 8 and Group 9 ($t = 4.46, p < 0.001$), and Group 9 and Group 10 ($t = 4.01, p < 0.001$). The difference between Group 10 and Group 11 is significant at the 0.05 level ($t = 2.21, p = 0.0399$). No significant difference is found between Group 11 and Group 12 ($t = -0.16, p = 0.8779$). Significant differences are observed between Group 12 and Group 13 ($t = 3.68, p = 0.0016$). No significant differences are found between Group 13 and Group 14 ($t = 0.70, p = 0.4947$). A significant difference is observed between Group 14 and Group 15 ($t = 2.68, p = 0.0147$). Finally, no significant difference is found between Group 15 and Group 16 ($t = -0.78, p = 0.4439!$).

These findings suggest nuanced variations in the dependent variable across different groups, highlighting specific pairs where the differences are statistically significant. The pattern of significance indicates that:

The most substantial changes occur in the earlier groups (1 through 5), with highly significant differences between each pair. As the group numbers increase (representing an increase in the number of threads), the differences between adjacent groups become less consistently significant. There is a pattern of alternating significance and non-significance in the later groups, suggesting that performance improvements may plateau or become more incremental with higher thread counts. The lack of significant difference between the last two groups (15 and 16) might indicate that increasing the thread count beyond a certain point may not yield substantial performance improvements.

This pattern supports the hypothesis that as the number of threads increases, the difference between the values becomes less statistically significant, particularly in the higher thread count ranges. This could be due to factors such as diminishing returns from parallelization, potential overhead from thread management, or hardware limitations.

The one-way ANOVA test on the impact of threads on objective function

resulted in a $F-$statistic of approximately 0.1565 and a p-value of approximately 0.999. The high p-value suggests that there is no significant difference in the objective function value across different numbers of threads, meaning the number of threads does not have a statistically significant impact on the value based on this dataset.

In conclusion, increasing the number of threads has a statistically significant impact on the runtime, with no impact on the objective function value. Increasing from 1 to 10 threads and upwards shows no statistical difference, so increases beyond 10 may have diminishing returns.

A.5.2 Statistical testing of Thread Tuning for Size 2 instances

Threads	Mean Runtime (s)	Standard Deviation (s)
1	365.47	25.97
2	292.14	18.58
3	188.49	12.20
4	170.20	11.38
5	129.59	7.57
6	123.61	7.20
7	106.63	5.36
8	103.74	5.71
9	93.61	5.02
10	87.72	5.73
11	85.97	5.23
12	85.81	6.15
13	77.93	4.07
14	79.09	4.64
15	80.94	4.38
16	84.00	5.54

Table A.37: Thread performance comparison for Size 2 instances

Source	F Value	Num DF	Den DF	Pr > F
Threads	3086.2645	15.0000	285.0000	0.0000

Table A.38: Summary of Repeated Measures ANOVA

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
1	2	-73329.25	0.0	-84524.2403	-62134.2597	True
1	3	-176978.8	0.0	-188173.7903	-165783.8097	True
1	4	-195266.15	0.0	-206461.1403	-184071.1597	True
1	5	-235877.25	0.0	-247072.2403	-224682.2597	True
1	6	-241857.8	0.0	-253052.7903	-230662.8097	True
1	7	-258834.5	0.0	-270029.4903	-247639.5097	True
1	8	-261731.7	0.0	-272926.6903	-250536.7097	True
1	9	-271862.85	0.0	-283057.8403	-260667.8597	True
1	10	-277751.75	0.0	-288946.7403	-266556.7597	True
1	11	-279494.1	0.0	-290689.0903	-268299.1097	True
1	12	-279663.2	0.0	-290858.1903	-268468.2097	True
1	13	-287537.25	0.0	-298732.2403	-276342.2597	True
1	14	-286376.55	0.0	-297571.5403	-275181.5597	True
1	15	-284531.5	0.0	-295726.4903	-273336.5097	True
1	16	-281465.25	0.0	-292660.2403	-270270.2597	True
2	3	-103649.55	0.0	-114844.5403	-92454.5597	True
2	4	-121936.9	0.0	-133131.8903	-110741.9097	True
2	5	-162548.0	0.0	-173742.9903	-151353.0097	True
2	6	-168528.55	0.0	-179723.5403	-157333.5597	True
2	7	-185505.25	0.0	-196700.2403	-174310.2597	True
2	8	-188402.45	0.0	-199597.4403	-177207.4597	True
2	9	-198533.6	0.0	-209728.5903	-187338.6097	True
2	10	-204422.5	0.0	-215617.4903	-193227.5097	True
2	11	-206164.85	0.0	-217359.8403	-194969.8597	True
2	12	-206333.95	0.0	-217528.9403	-195138.9597	True
2	13	-214208.0	0.0	-225402.9903	-203013.0097	True
2	14	-213047.3	0.0	-224242.2903	-201852.3097	True
2	15	-211202.25	0.0	-222397.2403	-200007.2597	True
2	16	-208136.0	0.0	-219330.9903	-196941.0097	True
3	4	-18287.35	0.0	-29482.3403	-7092.3597	True
3	5	-58898.45	0.0	-70093.4403	-47703.4597	True
3	6	-64879.0	0.0	-76073.9903	-53684.0097	True
3	7	-81855.7	0.0	-93050.6903	-70660.7097	True
3	8	-84752.9	0.0	-95947.8903	-73557.9097	True
3	9	-94884.05	0.0	-106079.0403	-83689.0597	True
3	10	-100772.95	0.0	-111967.9403	-89577.9597	True
3	11	-102515.3	0.0	-113710.2903	-91320.3097	True
3	12	-102684.4	0.0	-113879.3903	-91489.4097	True
3	13	-110558.45	0.0	-121753.4403	-99363.4597	True
3	14	-109397.75	0.0	-120592.7403	-98202.7597	True
3	15	-107552.7	0.0	-118747.6903	-96357.7097	True
3	16	-104486.45	0.0	-115681.4403	-93291.4597	True

Table A.39: Tukey HSD test results for threads Size 2 Part 1

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
4	5	-40611.1	0.0	-51806.0903	-29416.1097	True
4	6	-46591.65	0.0	-57786.6403	-35396.6597	True
4	7	-63568.35	0.0	-74763.3403	-52373.3597	True
4	8	-66465.55	0.0	-77660.5403	-55270.5597	True
4	9	-76596.7	0.0	-87791.6903	-65401.7097	True
4	10	-82485.6	0.0	-93680.5903	-71290.6097	True
4	11	-84227.95	0.0	-95422.9403	-73032.9597	True
4	12	-84397.05	0.0	-95592.0403	-73202.0597	True
4	13	-92271.1	0.0	-103466.0903	-81076.1097	True
4	14	-91110.4	0.0	-102305.3903	-79915.4097	True
4	15	-89265.35	0.0	-100460.3403	-78070.3597	True
4	16	-86199.1	0.0	-97394.0903	-75004.1097	True
5	6	-5980.55	0.8974	-17175.5403	5214.4403	False
5	7	-22957.25	0.0	-34152.2403	-11762.2597	True
5	8	-25854.45	0.0	-37049.4403	-14659.4597	True
5	9	-35985.6	0.0	-47180.5903	-24790.6097	True
5	10	-41874.5	0.0	-53069.4903	-30679.5097	True
5	11	-43616.85	0.0	-54811.8403	-32421.8597	True
5	12	-43785.95	0.0	-54980.9403	-32590.9597	True
5	13	-51660.0	0.0	-62854.9903	-40465.0097	True
5	14	-50499.3	0.0	-61694.2903	-39304.3097	True
5	15	-48654.25	0.0	-59849.2403	-37459.2597	True
5	16	-45588.0	0.0	-56782.9903	-34393.0097	True
6	7	-16976.7	0.0	-28171.6903	-5781.7097	True
6	8	-19873.9	0.0	-31068.8903	-8678.9097	True
6	9	-30005.05	0.0	-41200.0403	-18810.0597	True
6	10	-35893.95	0.0	-47088.9403	-24698.9597	True
6	11	-37636.3	0.0	-48831.2903	-26441.3097	True
6	12	-37805.4	0.0	-49000.3903	-26610.4097	True
6	13	-45679.45	0.0	-56874.4403	-34484.4597	True
6	14	-44518.75	0.0	-55713.7403	-33323.7597	True
6	15	-42673.7	0.0	-53868.6903	-31478.7097	True
6	16	-39607.45	0.0	-50802.4403	-28412.4597	True
7	8	-2897.2	0.9999	-14092.1903	8297.7903	False
7	9	-13028.35	0.0072	-24223.3403	-1833.3597	True
7	10	-18917.25	0.0	-30112.2403	-7722.2597	True
7	11	-20659.6	0.0	-31854.5903	-9464.6097	True
7	12	-20828.7	0.0	-32023.6903	-9633.7097	True
7	13	-28702.75	0.0	-39897.7403	-17507.7597	True
7	14	-27542.05	0.0	-38737.0403	-16347.0597	True
7	15	-25697.0	0.0	-36891.9903	-14502.0097	True
7	16	-22630.75	0.0	-33825.7403	-11435.7597	True

Table A.40: Tukey HSD test results for threads Size 2 Part 2

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
8	9	-10131.15	0.1274	-21326.1403	1063.8403	False
8	10	-16020.05	0.0001	-27215.0403	-4825.0597	True
8	11	-17762.4	0.0	-28957.3903	-6567.4097	True
8	12	-17931.5	0.0	-29126.4903	-6736.5097	True
8	13	-25805.55	0.0	-37000.5403	-14610.5597	True
8	14	-24644.85	0.0	-35839.8403	-13449.8597	True
8	15	-22799.8	0.0	-33994.7903	-11604.8097	True
8	16	-19733.55	0.0	-30928.5403	-8538.5597	True
9	10	-5888.9	0.9084	-17083.8903	5306.0903	False
9	11	-7631.25	0.5828	-18826.2403	3563.7403	False
9	12	-7800.35	0.5435	-18995.3403	3394.6403	False
9	13	-15674.4	0.0002	-26869.3903	-4479.4097	True
9	14	-14513.7	0.0011	-25708.6903	-3318.7097	True
9	15	-12668.65	0.0108	-23863.6403	-1473.6597	True
9	16	-9602.4	0.1911	-20797.3903	1592.5903	False
10	11	-1742.35	1.0	-12937.3403	9452.6403	False
10	12	-1911.45	1.0	-13106.4403	9283.5403	False
10	13	-9785.5	0.1669	-20980.4903	1409.4903	False
10	14	-8624.8	0.3604	-19819.7903	2570.1903	False
10	15	-6779.75	0.7686	-17974.7403	4415.2403	False
10	16	-3713.5	0.9989	-14908.4903	7481.4903	False
11	12	-169.1	1.0	-11364.0903	11025.8903	False
11	13	-8043.15	0.4875	-19238.1403	3151.8403	False
11	14	-6882.45	0.7482	-18077.4403	4312.5403	False
11	15	-5037.4	0.9748	-16232.3903	6157.5903	False
11	16	-1971.15	1.0	-13166.1403	9223.8403	False
12	13	-7874.05	0.5264	-19069.0403	3320.9403	False
12	14	-6713.35	0.7814	-17908.3403	4481.6403	False
12	15	-4868.3	0.9816	-16063.2903	6326.6903	False
12	16	-1802.05	1.0	-12997.0403	9392.9403	False
13	14	1160.7	1.0	-10034.2903	12355.6903	False
13	15	3005.75	0.9999	-8189.2403	14200.7403	False
13	16	6072.0	0.8857	-5122.9903	17266.9903	False
14	15	1845.05	1.0	-9349.9403	13040.0403	False
14	16	4911.3	0.98	-6283.6903	16106.2903	False
15	16	3066.25	0.9999	-8128.7403	14261.2403	False

Table A.41: Tukey HSD test results for threads Size 2 Part 3

The results of the Tukey Honestly Significant Difference (HSD) test reveal significant differences in the mean runtime values across different thread groups (Groups 1 to 16). Specifically:

There is a statistically significant difference ($p < 0.05$) in the mean runtime between Group 1 (single thread) and all other groups (Groups 2 to 16). Significant differences exist among most pairs of groups, with some notable exceptions: a) There is no statistically significant difference between Group 5 and Group 6 ($p = 0.8974$). b) Group 7 does not significantly differ from Group 8 ($p = 0.9999$). c) Group 8 does not significantly differ from Group 9 ($p = 0.1274$). d) Groups 9, 10, 11, and 12 do not significantly differ from each other (all $p > 0.05$). e) Groups 10 through 16 show no significant differences among themselves (all $p > 0.05$). The largest mean differences are observed between Group 1 and the higher-numbered groups, with the difference increasing as the group number increases. The maximum mean difference is observed between Group 1 and Group 13 (-287,537.25). From Group 9 onwards, the differences between consecutive groups become smaller and statistically insignificant in most cases. The improvement in runtime is most pronounced in the early increases in thread count (from 1 to 8 threads), with statistically significant improvements observed at each step. After 9 threads, the improvements in runtime become less substantial and often not statistically significant, indicating a plateau in performance gains.

These findings suggest that the number of threads significantly influences the mean runtime, with single-threaded execution (Group 1) showing significant differences compared to all multi-threaded executions. The most substantial and statistically significant improvements occur when increasing threads from 1 to 9. Beyond 9 threads, while there are still some numerical improvements in runtime, these differences are often not statistically significant. This pattern indicates that there is a point of diminishing returns in adding more threads, likely due to factors such as increased overhead in thread management or limitations in the underlying hardware or workload parallelizability. The optimal thread count for this particular workload

appears to be around 9-13 threads, as further increases do not yield statistically significant improvements in runtime.

Cohen's d effect sizes:

Comparison	**Cohen's d**
Group 1 vs. Group 2	3.25
Group 1 vs. Group 3	8.72
Group 1 vs. Group 4	9.74
Group 1 vs. Group 5	12.33
Group 1 vs. Group 6	12.69
Group 1 vs. Group 7	13.80
Group 1 vs. Group 8	13.92
Group 1 vs. Group 9	14.53
Group 1 vs. Group 10	14.77
Group 1 vs. Group 11	14.92
Group 1 vs. Group 12	14.82
Group 1 vs. Group 13	15.47
Group 1 vs. Group 14	15.35
Group 1 vs. Group 15	15.28
Group 1 vs. Group 16	14.99

Table A.42: Cohen's d effect sizes for pairwise comparisons for Size 2

We get similar results as before, with the strongest effects when compared to running the single-threaded GRASP peaking around 13-14 threads.

Pair	T**-Value**	p**-value**
1 vs. 2	24.36	8.60×10^{-16}
2 vs. 3	44.61	1.07×10^{-20}
3 vs. 4	15.08	5.02×10^{-12}
4 vs. 5	33.05	2.97×10^{-18}
5 vs. 6	7.82	2.35×10^{-7}
6 vs. 7	17.68	2.97×10^{-13}
7 vs. 8	4.17	5.20×10^{-4}
8 vs. 9	18.67	1.11×10^{-13}
9 vs. 10	12.46	1.37×10^{-10}
10 vs. 11	2.96	8.12×10^{-3}
11 vs. 12	0.33	0.7447
12 vs. 13	12.16	2.07×10^{-10}
13 vs. 14	-2.68	0.0150
14 vs. 15	-3.75	1.36×10^{-3}
15 vs. 16	-3.29	3.89×10^{-3}

Table A.43: Paired t-tests results for Size 2

All pairs up to and including 9 vs. 10 show highly significant differences ($p < 0.001$), indicating substantial performance improvements with each increase in thread count up to 10 threads. The pair 10 vs. 11 shows a significant difference ($p < 0.01$), suggesting that there is still a measurable improvement when moving from 10 to 11 threads. The pair 11 vs. 12 shows no significant difference ($p = 0.7447$), indicating that the performance improvement plateaus at this point. Interestingly, the pair 12 vs. 13 shows a highly significant difference again ($p < 0.001$), suggesting a performance jump when moving to 13 threads. For pairs 13 vs. 14, 14 vs. 15, and 15 vs. 16, we see significant differences ($p < 0.05$), but with negative t-values. This suggests that performance actually decreases slightly with these additional threads.

This analysis suggests that the optimal thread count for this particular workload is likely 13-14, as it shows the last significant performance improvement before we start to see decreases. The slight performance decreases beyond 14 threads might be due to increased overhead from managing additional threads or resource contention.

The one-way ANOVA test on the impact of threads on objective function resulted in a $F-$statistic of approximately 0.1326 and a p-value of approximately 0.999. The high p-value suggests that there is no significant difference in the objective function value across different numbers of threads, meaning the number of threads does not have a statistically significant impact on the value based on this dataset, repeating the expected results from Size 1, so it is safe to say that using more threads has no impact, positive or negative, on the quality of the solution.

In conclusion, increasing the number of threads has a statistically significant impact on the runtime, with no impact on the objective function value. Increasing from 1 to 9 threads and upwards shows no statistical difference, and increases above 14 threads actually have negative performance, so 14 threads should be the ideal number to be used.

A.5.3 Statistical testing of Parameter Tuning for Size 1 instances

The One-Way ANOVA for the runtime of the GRASP Metaheuristic across different levels of iterations yielded an F-statistic of 8226.66 and a p-value of 0.0, indicating statistically significant differences in runtime across the iteration levels. For the α_a and α_l parameters, the runtime analysis resulted in F-statistics of 0.01 and 0.10, with p-values of 0.9985 and 0.9580 respectively, suggesting no significant impact on runtime. The analysis for the objective function value resulted in an F-statistic of 11.42 and a p-value of 2.652×10^{-7} for iterations, 4.83 and 0.0025 for α_a, and 7.09 and 1.089×10^{-4} for α_l, all showing statistically significant differences in solution

quality across these parameter levels.

Factorial ANOVA extends the concept of One-Way ANOVA by allowing for the examination of the effects of two or more independent variables (factors) simultaneously on a single dependent variable. It enables the analysis of not only the main effects of each factor but also the interaction effects between them. In the context of analyzing the performance of a metaheuristic algorithm, Factorial ANOVA allows us to understand not only how individual parameters (e.g., α_a, α_l and iterations) affect the algorithm's efficiency and solution quality but also how these parameters interact with each other to influence the outcomes. This comprehensive analysis is crucial for identifying optimal parameter settings, especially in complex systems where the interaction between parameters can significantly impact performance.

Effect	**Metric**	**F-Statistic**	**p-Value**
Time			
Main Effects - iters	Time	8347.26	< 0.000000
Main Effects - α_a	Time	0.43	0.735007
Main Effects - α_l	Time	4.18	0.006107
Interaction Effects (iters, α_a)	Time	0.85	0.572186
Interaction Effects (iters, α_l)	Time	0.94	0.493059
Interaction Effects (α_a, α_l)	Time	1.39	0.191076
Objective Value			
Main Effects - iters	Objective Value	11.64	< 0.0000002
Main Effects - α_a	Objective Value	5.07	0.001792
Main Effects - α_l	Objective Value	7.37	0.000075
Interaction Effects (iters, α_a)	Objective Value	0.14	0.998433
Interaction Effects (iters, α_l)	Objective Value	0.40	0.934506
Interaction Effects (α_a, α_l)	Objective Value	0.34	0.959751

Table A.44: Factorial ANOVA results for time and objective value for Size 1

The results show that the number of iterations had a very significant effect ($p < 0.000000$) on time, while α_a and α_l are not not significant. All interactions between parameters were not statistically significant. On the other hand, the number of iterations and both α parameters had a significant impact on objective values.

Detailed pairwise comparisons were conducted for both Time and Objective Function Value across different levels of iterations and α parameters identifying specific pairs that differ significantly. The Tukey HSD test results offer a comprehensive look at pairwise differences among the levels of iterations, highlighting significant disparities that inform the optimization of algorithm parameters.

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	3479.96	0.0	3298.29	3661.62	True
25	75	7076.15	0.0	6894.48	7257.82	True
25	100	10479.74	0.0	10298.08	10661.41	True
50	75	3596.19	0.0	3414.53	3777.86	True
50	100	6999.79	0.0	6818.12	7181.45	True
75	100	3403.59	0.0	3221.93	3585.26	True

Table A.45: Tukey HSD test results for iterations on runtime for Size 1

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	-3178.83	0.0045	-5611.92	-745.75	True
25	75	-3916.19	0.0002	-6349.28	-1483.11	True
25	100	-5336.96	0.0	-7770.04	-2903.87	True
50	75	-737.36	0.8632	-3170.45	1695.72	False
50	100	-2158.13	0.1026	-4591.21	274.96	False
75	100	-1420.76	0.4356	-3853.85	1012.32	False

Table A.46: Tukey HSD test results for iterations on objective value for Size 1

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	61.50	0.9991	-1084.63	1207.63	False
0.1	0.5	10.82	1.0	-1135.31	1156.95	False
0.1	0.7	-11.87	1.0	-1158.00	1134.26	False
0.3	0.5	-50.68	0.9995	-1196.81	1095.45	False
0.3	0.7	-73.37	0.9984	-1219.50	1072.76	False
0.5	0.7	-22.69	1.0	-1168.82	1123.44	False

Table A.47: Tukey HSD test results for α_a on runtime

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	3074.41	0.0077	604.60	5544.23	True
0.1	0.5	3046.51	0.0085	576.69	5516.32	True
0.1	0.7	2779.45	0.0202	309.63	5249.27	True
0.3	0.5	-27.91	1.0	-2497.72	2441.91	False
0.3	0.7	-294.96	0.9899	-2764.78	2174.85	False
0.5	0.7	-267.06	0.9925	-2736.87	2202.76	False

Table A.48: Tukey HSD test results for α_a on objective value for Size 1

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	-113.53	0.9942	-1259.40	1032.35	False
0.1	0.5	-32.23	0.9999	-1178.10	1113.65	False
0.1	0.7	129.78	0.9914	-1016.10	1275.65	False
0.3	0.5	81.30	0.9978	-1064.58	1227.18	False
0.3	0.7	243.30	0.9474	-902.58	1389.18	False
0.5	0.7	162.00	0.9835	-983.88	1307.88	False

Table A.49: Tukey HSD test results for α_l on runtime for Size 1

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	-3296.43	0.0033	-5753.47	-839.39	True
0.1	0.5	-3220.80	0.0043	-5677.84	-763.76	True
0.1	0.7	-4030.83	0.0002	-6487.86	-1573.79	True
0.3	0.5	75.63	0.9998	-2381.41	2532.67	False
0.3	0.7	-734.39	0.8680	-3191.43	1722.64	False
0.5	0.7	-810.03	0.8308	-3267.06	1647.01	False

Table A.50: Tukey HSD test results for α_l on objective value for Size 1

For the runtime of the algorithm, the test results show significant differences between all pairs of groups (p-adj = 0.0 for all comparisons). This indicates that the mean time is significantly different between each pair of groups tested. The magnitude of the mean difference (meandiff) increases as the difference between group numbers increases. For example, the difference between groups of 25 iterations and 50 iterations is smaller than the difference between groups of 25 iterations and 100 iterations. This suggests a trend where groups with higher iterations have significantly longer times. All comparisons were found to be significant, indicating that as the group number increases, so does the time, and these differences are not due to random chance.

Unlike the time, not all group comparisons for the objective function value are significant. Significant differences are observed in some pairs (e.g., 25 vs 50, 25 vs 75, 25 vs 100), while others show no significant difference. The significant differences do not follow a clear linear trend as observed with the time. This implies that the objective function value measure does not consistently increase or decrease with the group numbers in a manner that is statistically significant across all comparisons. The significant negative meandiff values (e.g., 25 vs 100, -5336.96) suggest that certain groups have significantly lower values compared to others. Specifically, group 25 tends to have higher value scores compared to other groups, indicating a decrease in value scores as the group number increases. However, when comparing 75 vs 100

iterations, the decrease is no longer statistically significant.

The time shows a clear and significant increase in values as the group number increases, suggesting a strong and consistent difference across all group comparisons.

The value, however, presents a more complex picture with significant differences only in certain group comparisons, indicating that while there are differences in value scores between specific groups, these differences are not uniformly observed across all groups.

α_a and α_l have no significant effect whatsoever on runtime, for any value they may take. Significant differences are observed between $\alpha_a = 0.1$ and all other values (0.3, 0.5, and 0.7). These comparisons have p-values < 0.05 and are rejecting the null hypothesis. The mean differences are positive, indicating that $\alpha_a = 0.1$ yields higher (worse) objective values compared to the other levels. There are no significant differences among values of 0.3, 0.5, and 0.7. Significant differences are observed between $\alpha_l = 0.1$ and all other values (0.3, 0.5, and 0.7). These comparisons have p-values < 0.05 and again reject the null hypothesis. The mean differences are negative, indicating that $\alpha_l = 0.1$ yields lower (better) objective values compared to the other levels. There are no significant differences among values of 0.3, 0.5, and 0.7.

These results suggest that while neither α_a nor α_l significantly affect runtime, they do impact the solution quality. Specifically, $\alpha_a = 0.1$ leads to worse solutions, while $\alpha_l = 0.1$ leads to better solutions compared to their respective higher values. For both parameters, values of 0.3, 0.5, and 0.7 perform similarly to each other in terms of solution quality.

Therefore, for instances of Size 1, the tuned parameters are set to the following values: iterations set to 100, although they increase time, they also improve objective function value (marginally when set to 100 iterations), but the time penalty is negligible; α_a set to 0.3 and α_l set to 0.1.

A.5.4 Statistical testing of Parameter Tuning for Size 2 instances

The One-Way ANOVA for the runtime of the GRASP Metaheuristic across different levels of iterations yielded an F-statistic of 5219.33 and a p-value of 0.0, indicating statistically significant differences in runtime across the iteration levels. For the α_a and α_l parameters, the runtime analysis resulted in F-statistics of 0.86 and 2.38, with p-values of 0.4637 and 0.0683 respectively, suggesting no significant impact on runtime. The analysis for the objective function value resulted in an F-statistic of 6.81 and a p-value of 1.602×10^{-4} for iterations, 1.46 and 0.2233 for α_a, and 0.51 and 0.6765 for α_l, showing statistically significant differences in solution quality across iteration levels, but not for the α parameters.

Effect	**Metric**	**F-Statistic**	**p-Value**
Time			
Main Effects - iters	Time	9562.34	< 0.000000
Main Effects - α_a	Time	40.01	1.364×10^{-23}
Main Effects - α_l	Time	110.64	4.445×10^{-57}
Interaction Effects (iters, α_a)	Time	3.76	1.281×10^{-4}
Interaction Effects (iters, α_l)	Time	7.91	4.337×10^{-11}
Interaction Effects (α_a, α_l)	Time	0.58	0.811264
Objective Value			
Main Effects - iters	Objective Value	6.58	0.000223
Main Effects - α_a	Objective Value	1.45	0.227379
Main Effects - α_l	Objective Value	0.51	0.678434
Interaction Effects (iters, α_a)	Objective Value	0.33	0.963511
Interaction Effects (iters, α_l)	Objective Value	0.15	0.998117
Interaction Effects (α_a, α_l)	Objective Value	0.12	0.999104

Table A.51: Factorial ANOVA results for time and objective value for Size 2

The results show that the number of iterations had a very significant effect ($p < 0.000000$) on time, while α_a and α_l also showed significant effects, contrary to the one-way ANOVA results. Interactions between iterations and both α parameters were statistically significant for time, but not between the α parameters themselves. For objective function values, only the number of iterations showed a significant impact, while neither the α parameters nor any interactions were significant. Detailed pairwise comparisons were conducted for both Time and Objective Function Value across different levels of iterations and α parameters identifying specific pairs that differ significantly. The Tukey HSD test results offer a comprehensive look at pairwise differences among the levels of iterations, highlighting significant disparities that inform the optimization of algorithm parameters.

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	38422.49	0.0	35962.50	40882.47	True
25	75	76418.65	0.0	73958.66	78878.64	True
25	100	113297.45	0.0	110837.46	115757.44	True
50	75	37996.16	0.0	35536.18	40456.15	True
50	100	74874.96	0.0	72414.98	77334.95	True
75	100	36878.80	0.0	34418.81	39338.79	True

Table A.52: Tukey HSD test results for iterations on runtime for Size 2

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	-4828.01	0.1775	-10949.99	1293.98	False
25	75	-7791.22	0.0061	-13913.20	-1669.24	True
25	100	-10179.63	0.0001	-16301.61	-4057.64	True
50	75	-2963.21	0.5973	-9085.20	3158.77	False
50	100	-5351.62	0.1107	-11473.60	770.36	False
75	100	-2388.41	0.7466	-8510.39	3733.58	False

Table A.53: Tukey HSD test results for iterations on objective value for Size 2

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	3237.18	0.908	-9189.15	15663.51	False
0.1	0.5	5409.20	0.6765	-7017.13	17835.53	False
0.1	0.7	7363.28	0.4223	-5063.05	19789.61	False
0.3	0.5	2172.02	0.9696	-10254.31	14598.35	False
0.3	0.7	4126.10	0.8278	-8300.23	16552.43	False
0.5	0.7	1954.08	0.9775	-10472.25	14380.41	False

Table A.54: Tukey HSD test results for α_a on runtime for Size 2

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	4385.91	0.2635	-1812.25	10584.06	False
0.1	0.5	4347.86	0.271	-1850.29	10546.02	False
0.1	0.7	2908.76	0.6215	-3289.40	9106.91	False
0.3	0.5	-38.04	1.0	-6236.20	6160.11	False
0.3	0.7	-1477.15	0.9277	-7675.30	4721.00	False
0.5	0.7	-1439.11	0.9326	-7637.26	4759.05	False

Table A.55: Tukey HSD test results for α_a on objective value for Size 2

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	872.45	0.9979	-11509.53	13254.43	False
0.1	0.5	5081.68	0.7157	-7300.30	17463.66	False
0.1	0.7	11496.06	0.0797	-885.92	23878.03	False
0.3	0.5	4209.23	0.8175	-8172.75	16591.21	False
0.3	0.7	10623.61	0.1216	-1758.37	23005.58	False
0.5	0.7	6414.38	0.5413	-5967.60	18796.35	False

Table A.56: Tukey HSD test results for α_l on runtime for Size 2

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	790.05	0.9879	-5422.02	7002.12	False
0.1	0.5	2650.46	0.6903	-3561.61	8862.53	False
0.1	0.7	2135.49	0.8124	-4076.58	8347.56	False
0.3	0.5	1860.41	0.8674	-4351.66	8072.48	False
0.3	0.7	1345.44	0.9444	-4866.63	7557.51	False
0.5	0.7	-514.98	0.9966	-6727.04	5697.09	False

Table A.57: Tukey HSD test results for α_l on objective value for Size 2

For the runtime of the algorithm, the test results show significant differences between all pairs of groups (p-adj = 0.0 for all comparisons). This indicates that the

mean time is significantly different between each pair of groups tested. The magnitude of the mean difference (meandiff) increases as the difference between group numbers increases. For example, the difference between groups of 25 iterations and 50 iterations is smaller than the difference between groups of 25 iterations and 100 iterations. This suggests a trend where groups with higher iterations have significantly longer times. All comparisons were found to be significant, indicating that as the group number increases, so does the time, and these differences are not due to random chance.

For the objective function value, significant differences are observed only in some pairs (25 vs 75, 25 vs 100), while others show no significant difference. The significant differences do not follow a clear linear trend as observed with the time. This implies that the objective function value measure does not consistently increase or decrease with the group numbers in a manner that is statistically significant across all comparisons. The significant negative meandiff values (e.g., 25 vs 100, -10179.63) suggest that certain groups have significantly lower values compared to others. Specifically, group 25 tends to have higher value scores compared to groups 75 and 100, indicating a decrease in value scores as the group number increases from 25 to 75 or 100. However, when comparing 50 vs 75, 50 vs 100, or 75 vs 100 iterations, the decreases are no longer statistically significant.

The time shows a clear and significant increase in values as the group number increases, suggesting a strong and consistent difference across all group comparisons. The value, however, presents a more complex picture with significant differences only in certain group comparisons, indicating that while there are differences in value scores between specific groups, these differences are not uniformly observed across all groups. Unlike in the previous results, α_a and α_l show no significant effect on either runtime or objective function value for any value they may take. All comparisons for both parameters across all levels have p-values > 0.05, failing to reject the null hypothesis. This suggests that changes in α_a and α_l do not lead to statistically significant changes in either runtime or solution quality.

These results suggest that while the number of iterations significantly affects both runtime and solution quality (to a certain extent), neither α_a nor α_l have a statistically significant impact on these metrics. This is in contrast to the factorial ANOVA results, which showed significant main effects for α_a and α_l on runtime. This discrepancy might be due to the different nature of the tests or potential interactions that are captured in the factorial ANOVA but not in the pairwise comparisons.

Therefore, for these instances of Size 2, the tuned parameters could be set as follows: iterations set to 75 or 100, as they improve objective function value significantly compared to 25 iterations, although they also increase time. The choice between 75 and 100 iterations would depend on the specific trade-off between solution quality and runtime that is acceptable for the problem at hand. As for α_a and α_l, given that they do not show statistically significant effects on either runtime or objective function value, their values could be set based on other considerations or left at default values, repeating the values for Size 1 of 0.3 and 0.1.

It is important to note that while the statistical tests do not show significant differences for α_a and α_l, there might still be practical differences that could be relevant in specific contexts. Additionally, the interaction effects observed in the factorial ANOVA for runtime suggest that the impact of these parameters might be more complex and intertwined with the number of iterations than the pairwise comparisons can reveal.

Future work might involve more detailed analysis of these interactions, or exploration of a wider range of values for α_a and α_l to see if significant effects emerge under different conditions. It might also be valuable to consider other performance metrics or to analyze the algorithm's behavior on different types of problem instances to get a more comprehensive understanding of the parameters' impacts.

A.6 Algorithms' graphical examples

A.6.1 The p-dispersion algorithm

Let us assume an instance of the problem with the configuration shown in the plot, with $|K| = 3$ to make the problem easier to follow. We have 3 center types along with their lower and upper bounds of centers to be used, figures and colors representing each type, along with 15 BUs, 14 centers and $p = 9$:

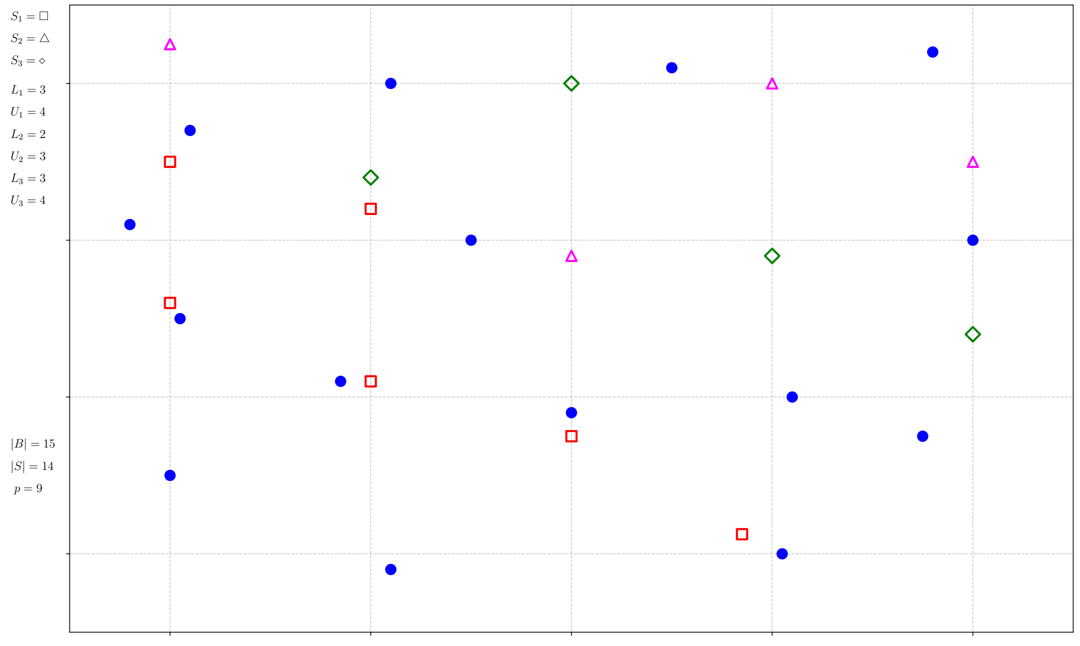


Figure A.1: Illustrative instance of the problem for p-dispersion algorithm

The location algorithm will solve $|K|$ p-dispersion subproblems with the heuristic presented in Algorithm 2. In this case, it will solve 3 subproblems, with the p of each problem set to the lower bound of the center type that it is trying to solve for, taking into account only the centers of that type, as follows:

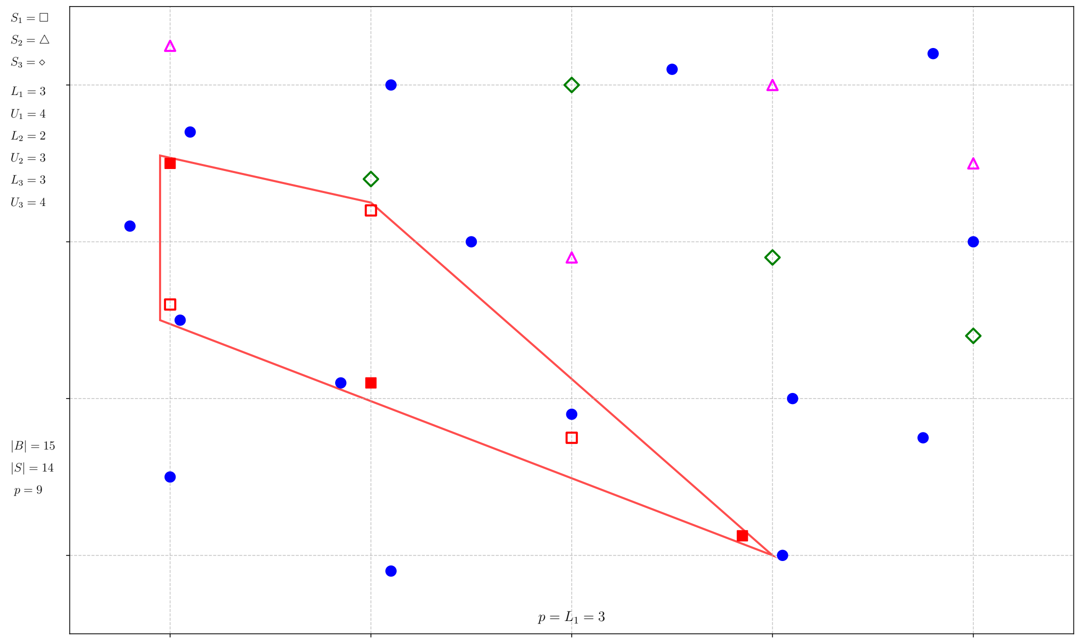


Figure A.2: Illustrative instance of the problem for p-dispersion for $k = 1$

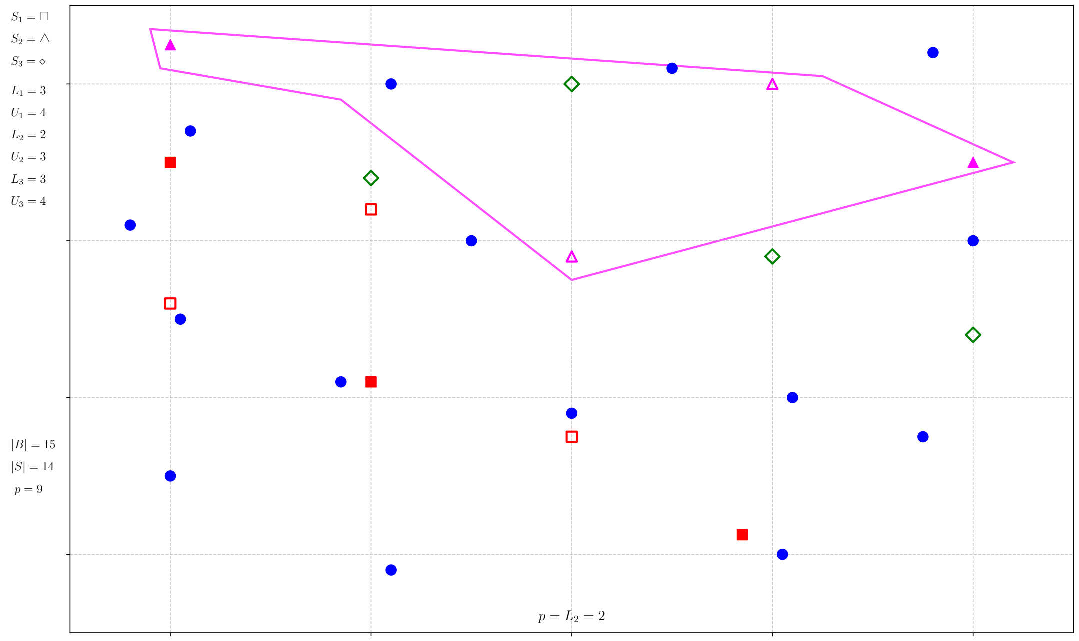


Figure A.3: Illustrative instance of the problem for p-dispersion for $k = 2$

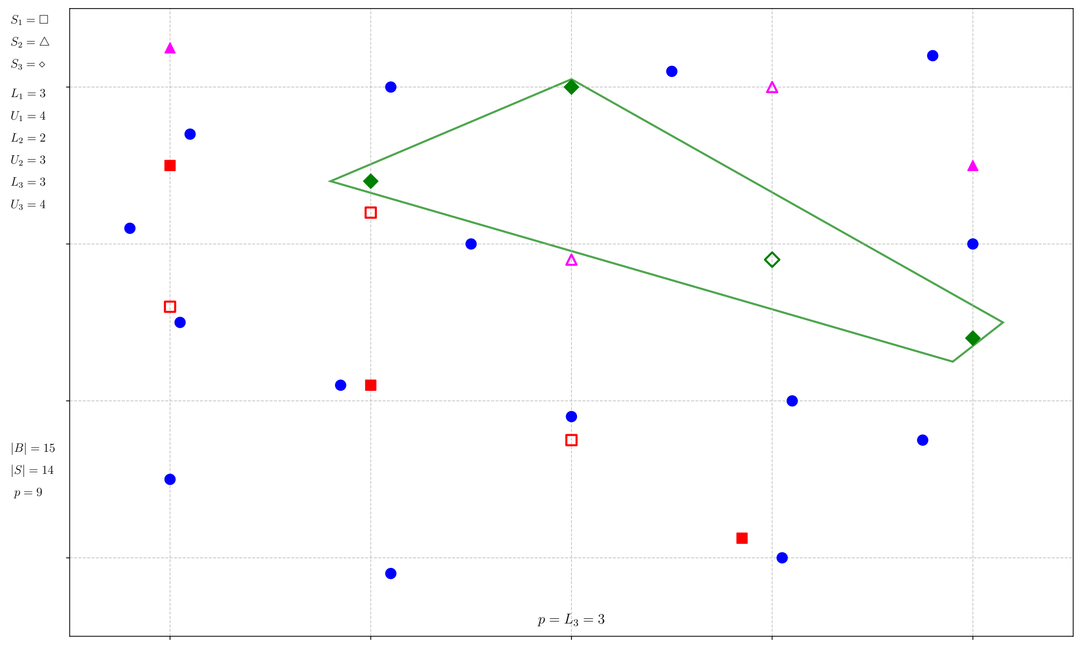


FIGURE A.4: Illustrative instance of the problem for p-dispersion for $k = 3$

After it is done with the subproblems, it combines the solutions into a larger set, and checks if the cardinality of the solution set is equal to the original p of the problem, which in this case is not, as it is missing one center. Therefore, it finally solves another p-dispersion problem, now taking all of the centers into account, adding centers to the solution only if their location does not produce an infeasible solution when taking into account U_k, in this case, it decides to add a new center of type 1, as shown in the following figures.

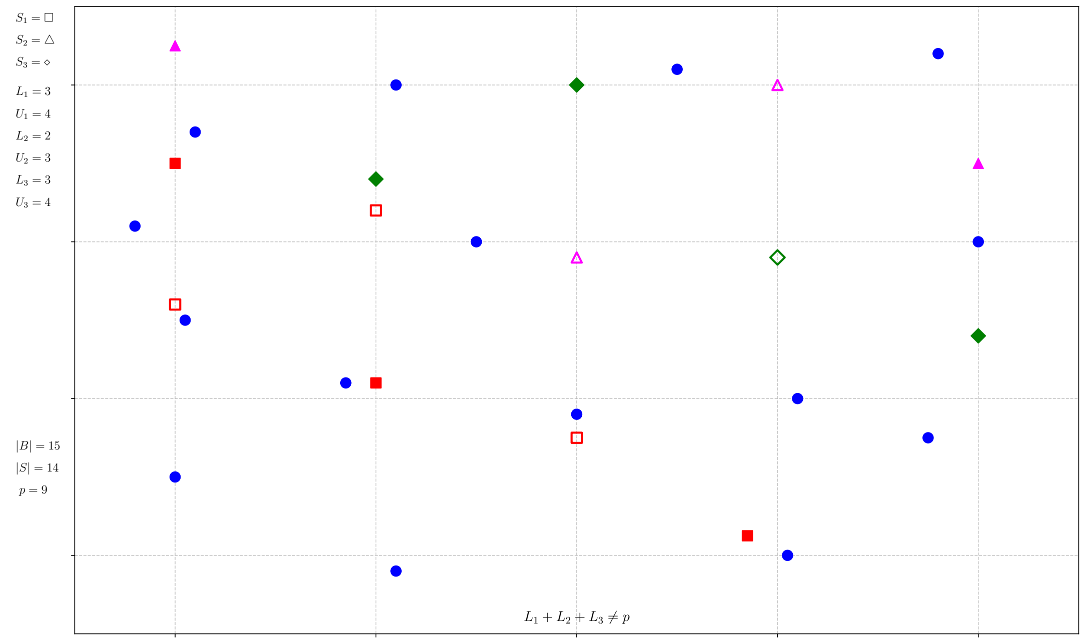


FIGURE A.5: Illustrative instance of the problem with the subproblems joined

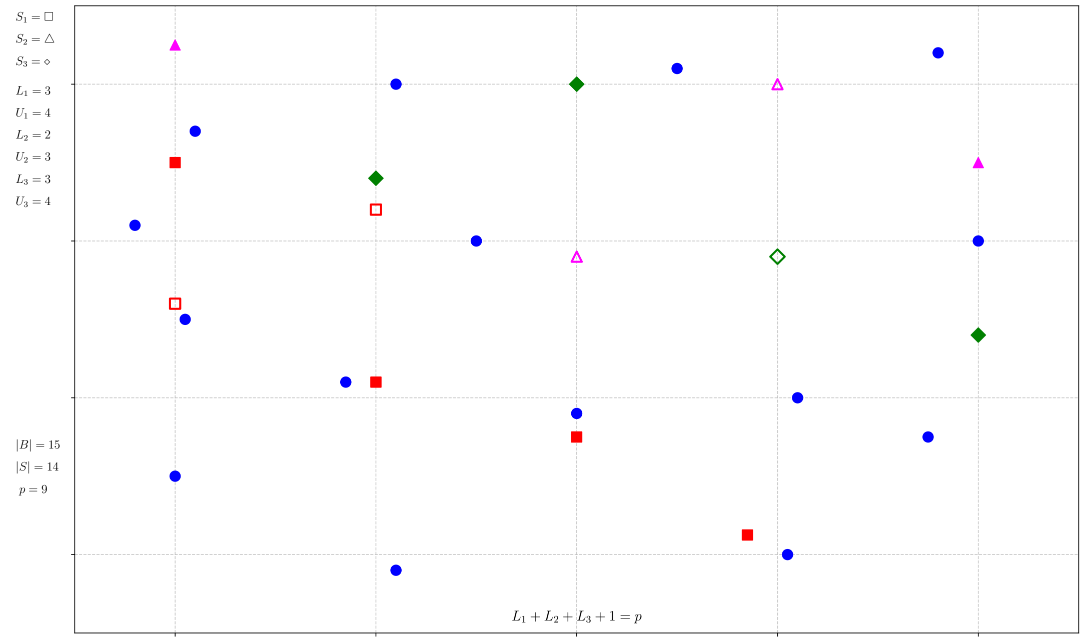


FIGURE A.6: Illustrative instance of the problem for p-dispersion as a whole

A.6.2 Allocation algorithm with opportunity cost

Let us assume an instance of the problem with 4 BUs and 2 centers, both opened, $Y_1 = 1, Y_2 = 1$, with the opportunity cost queue set as $\{1, 4, 2, 3\}$

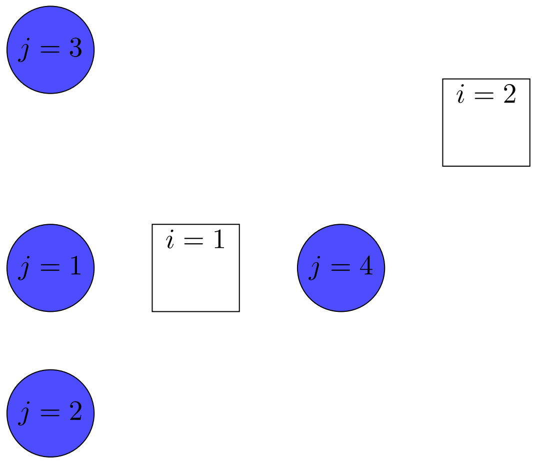


Figure A.7: Initial configuration

We dequeue the BU of $j = 1$ as it is the first element of the queue, and we assign it to its nearest center. We perform the calculation of the constraints of activity measures for $i = 1, j = 1$ and update the state of those constraints to the center, with $|M| = 3$. As a reminder, the description of the mathematical model is available in Section 3.3.

Let us assume the following values for $j = 1$: $v_1^1 = 10, v_1^2 = 30, v_1^3 = 23$ and the values of the constraints of the lower bounds for $i = 1$: $\mu_1^1 Y_1(1 - t^1) = 30, \mu_1^2 Y_1(1 - t^2) = 56, \mu_1^3 Y_1(1 - t^3) = 58$. We check if any constraint has reached the minimum value required, in this case, the center has not reached those values as seen in: $10 \leq 30$ for $m = 1$, $30 \leq 56$ for $m = 2$, $23 \leq 58$ for $m = 3$.

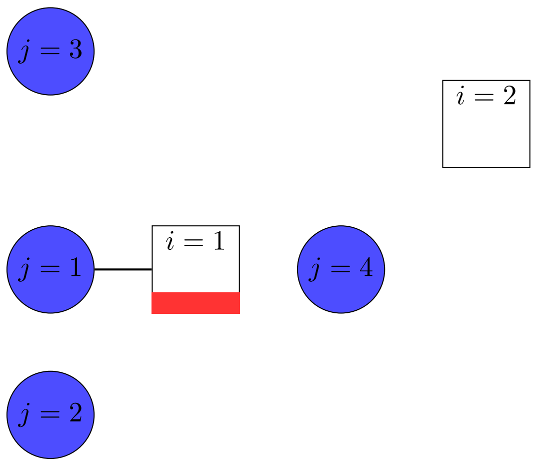


FIGURE A.8: First step with capacity constraints updated for $i = 1, j = 1$

We then dequeue the next BU, which is $j = 4$. The values of the activity measures that the BU provide are as follows: $j = 4$: $v_4^1 = 21, v_4^2 = 28, v_4^3 = 35$ and we update the values of $i = 1$: $10 + 21 > 30$ for $m = 1$, $30 + 28 > 56$ for $m = 2$, $23 + 35 > 58$ for $m = 3$. In this case, all activity measures have reached the minimum value required in the constraint, therefore we update the capacities of $i = 1$ and mark it as unavailable for future allocations.

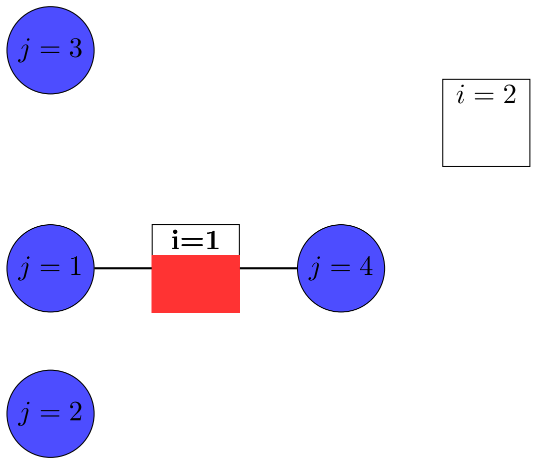


FIGURE A.9: Second step showing allocation from $j = 4$ to $i = 1$, unavailable.

We dequeue the next element, $j = 2$. In this case, its nearest center is $i = 1$, but it is unavailable to be allocated to, as it now complies with the constraints of the lower

bounds for the activity measures target, this forces other centers to be used, guiding the solution towards feasibility. As such, we allocate $j = 2$ to $i = 2$ and update the values for that center, with the values of the BU being: $v_2^1 = 15, v_2^2 = 37, v_2^3 = 25$, the constraints of the center being $\mu_1^1 Y_2(1-t^1) = 40, \mu_1^2 Y_2(1-t^2) = 50, \mu_1^3 Y_2(1-t^3) = 69$, with the state of the constraints being : $15 \leq 40$ for $m = 1$, $37 \leq 50$ for $m = 2$, $25 \leq 69$ for $m = 3$.

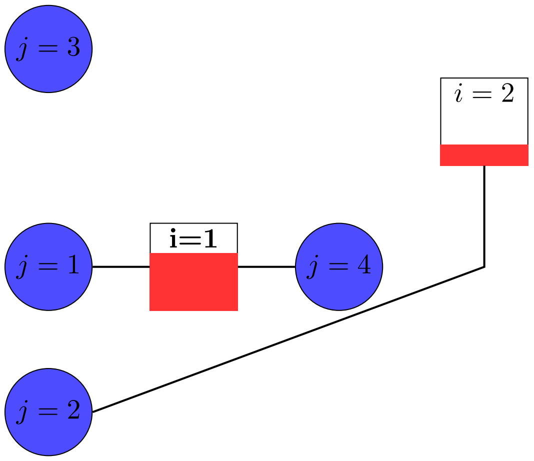


Figure A.10: Third step showing allocation from $j = 2$ to $i = 2$

Finally, we allocate $j = 3$ to $i = 2$, as it is the last element in the queue. We update the center capacities with the values from the BU $j = 3$: $v_3^1 = 26, v_3^2 = 30, v_3^3 = 48$, with the state of the constraints being : $15 + 26 > 40$ for $m = 1$, $37 + 30 > 50$ for $m = 2$, $25 + 48 > 69$ for $m = 3$. This makes $i = 2$ unavailable for future centers.

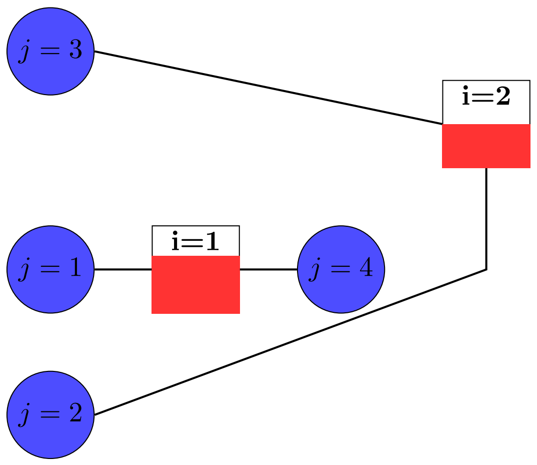


FIGURE A.11: Fourth step showing allocation from $j = 3$ to $i = 2$

The final assumption is that we introduce a fifth BU, $j = 5$, to illustrate what would happen in the case that all centers are unavailable due to their activity measures constraints.

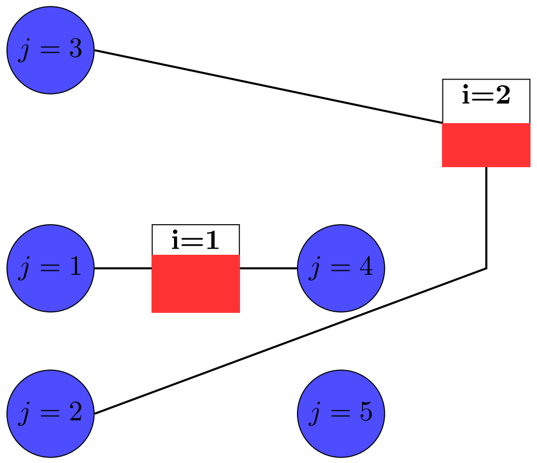


FIGURE A.12: Fifth step introducing $j = 5$

To handle unassigned BUs, we then switch to the risk threshold as the criterion of allocation, assigning the 5th BU to the center with the highest residual capacity in the risk threshold. By calculating the current used capacity in the risk threshold for each center, summing the values of the risks provided by each BU, we obtain the following results: we sum the risk values for $j = 1, j = 4$ as they are assigned

to $i = 1$, with a risk threshold of $\beta_1 = 120$, so the residual capacity for $i = 1$ is $r_1 = 40, r_4 = 15$, $120 - (40 + 15) = 65$, for $i = 2$ with a risk threshold of $\beta_2 = 150$ we find that the assignments are $j = 2$ and $j = 3$, $r_2 = 30, r_3 = 60$, then the capacity is $150 - (30 + 60) = 60$. In this case, $i = 1$ has the highest residual capacity in the risk threshold, so we assign $j = 5$ to it, and perform one last update on the state of the constraints, as shown in the increase of the red square representing the capacities used.

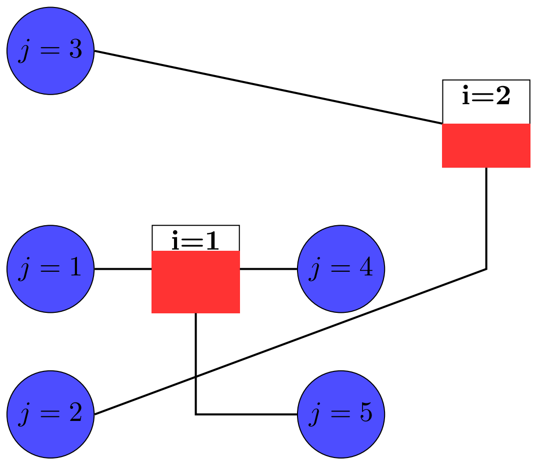


FIGURE A.13: Sixth step showing allocation from $j = 5$ to $i = 1$

Bibliography

G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483–485, New York, USA, 1967. Association for Computing Machinery.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2014.

G. Bruton, S. Khavul, D. Siegel, and M. Wright. New financial alternatives in seeding entrepreneurship: Microfinance, crowdfunding, and peer–to–peer innovations. *Entrepreneurship Theory and Practice*, 39(1):9–26, 2015.

J. Cano-Belmán, R. Z. Ríos-Mercado, and M. A. Salazar-Aguilar. Commercial territory design for a distribution firm with new constructive and destructive heuristics. *International Journal of Computational Intelligence Systems*, 5(1):126–147, 2012.

I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

E. Erkut and S. Neuman. Comparison of four models for dispersing facilities. *INFOR: Information Systems and Operational Research*, 29(2):68–86, 1991.

E. Erkut, Y. Ülküsal, and O. Yeniçerioğlu. A comparison of p-dispersion heuristics. *Computers & Operations Research*, 21(10):1103–1113, 1994.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

E. Fernández, J. Kalcsics, S. Nickel, and R. Z. Ríos-Mercado. A novel maximum dispersion territory design model arising in the implementation of the WEEE-directive. *Journal of the Operational Research Society*, 61(3):503–514, 2010.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL : `https://www.gurobi.com`.

P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

S. W. Hess, J. B. Weaver, H. J. Siegfeldt, J. N. Whelan, and P. A. Zitlau. Nonpartisan political redistricting by computer. *Operations Research*, 13(6):998–1006, 1965.

J. Kalcsics. Towards a unified territorial design approach - applications, algorithms and gis integration. *TOP*, 13:1–56, 2005.

J. Kalcsics and R. Z. Ríos-Mercado. Districting problems. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, chapter 25, pages 703–741. Springer, Cham, Switzerland, 2nd edition, 2019.

J. Kallrath. *Modeling Languages in Mathematical Optimization*. Springer, New York, USA, 2004.

R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

S. Khandker. Microfinance and poverty: Evidence using panel data from Bangladesh. *World Bank Economic Review*, 19:263–286, 02 2005.

S. Koranne. Hierarchical data format 5: Hdf5. In *Handbook of Open Source Tools*, pages 191–200. Springer, New York, USA, 2011.

G. Laporte, S. Nickel, and F. Saldanha Da Gama, editors. *Location Science*. Springer, Cham, Switzerland, 2019.

J. F. López, T. Ekin, F. Mendez Mediavilla, and J. A. Jimenez. Hybrid heuristic for dynamic location-allocation on micro-credit territory design. *Computacion y Sistemas*, 19(4):783–804, 2015.

J. F. López, T. Ekin, F. Mendez Mediavilla, and J. A. Jimenez. Risk-balanced territory design optimization for a micro finance institution. *Journal of Industrial & Management Optimization*, 16(2):741–758, 2020.

F. A. Medrano. The complete vertex p-center problem. *EURO Journal on Computational Optimization*, 8(3-4):327–343, 2020.

M. S. R. Monteiro. Bank-branch location and sizing under economies of scale. Master's thesis, Universidade do Porto, Portugal, 2005.

M. S. R. Monteiro and D. B. M. M. Fontes. Locating and sizing bank-branches by opening, closing or maintaining facilities. In H.-D. Haasis, H. Kopfer, and J. Schönberger, editors, *Operations Research Proceedings 2005*, pages 303–308, Berlin, Germany, 2006. Springer.

D. R. Quevedo-Orozco and R. Z. Ríos-Mercado. Improving the quality of heuristic solutions for the capacitated vertex p-center problem through iterated greedy local search and variable neighborhood descent. *Computers & Operations Research*, 62: 133–144, 2015.

S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

R. Z. Ríos-Mercado and J. F. Bard. An exact algorithm for designing optimal districts in the collection of waste electric and electronic equipment through an improved reformulation. *European Journal of Operational Research*, 276(1):259–271, 2019.

R. Z. Ríos-Mercado, editor. *Optimal Districting and Territory Design*, volume 284 of *International Series in Operations Research and Management Science*. Springer, Cham, Switzerland, 2020.

R. Z. Ríos-Mercado and H. J. Escalante. GRASP with path relinking for commercial districting. *Expert Systems with Applications*, 44:102–113, 2016.

R. Z. Ríos-Mercado and E. Fernández. A reactive GRASP for a commercial territory design problem with multiple balancing requirements. *Computers & Operations Research*, 36(3):755–776, 2009.

R. Z. Ríos-Mercado and J. F. López-Pérez. Commercial territory design planning with realignment and disjoint assignment requirements. *Omega*, 41(3):525–535, 2013.

R. Z. Ríos-Mercado, A. M. Álvarez Socarrás, A. Castrillón, and M. C. López-Locés. A location-allocation-improvement heuristic for districting with multiple-activity balancing constraints and p-median-based dispersion minimization. *Computers & Operations Research*, 126:105106, 2021.

M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and M. Cabrera-Ríos. New models for commercial territory design. *Networks and Spatial Economics*, 11(3):487–507, 2011.

M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and J. L. González-Velarde. GRASP strategies for a bi-objective commercial territory design problem. *Journal of Heuristics*, 19(2):179–200, 2013.

M. G. Sandoval, J. A. Díaz, and R. Z. Ríos-Mercado. An improved exact algorithm for a territory design problem with p-center-based dispersion minimization. *Expert Systems with Applications*, 146:113150, 2020.

J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*. Wiley, New York, USA, 2003.

E. D. Taillard. *Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem.* Graduate Texts in Operations Research. Springer International Publishing, Cham, Switzerland, 2023.

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
Acta de Titulación

Institución 190001
Programa 501363

642373/1847972/63753

52094
Acta Núm. ....

En la Ciudad de Monterrey, capital del Estado de Nuevo León, al día 25 del mes de Junio del año 2024, siendo las 12:00 horas, reunidos en las instalaciones de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA de la Universidad Autónoma de Nuevo León, los(as) profesores(as) DR. ROGER ZIRAHUEN RÍOS MERCADO, DRA. MARÍA ANGÉLICA SALAZAR AGUILAR, DR. VINCENT ANDRÉ LIONEL BOYER , quienes fueron designados por la dirección de la escuela o facultad para integrar el Comité de Titulación de EDUARDO SALAZAR TREVIÑO, quien cursó y aprobó todas y cada una de las unidades de aprendizaje del Programa Educativo de LICENCIATURA COMO INGENIERO EN TECNOLOGÍA DE SOFTWARE, tal como lo dispone la Ley Orgánica de la Universidad Autónoma de Nuevo León publicada en el Periódico Oficial el siete de junio de mil novecientos setenta y uno, y en su Título Quinto: De la titulación, Capítulo I, De la obtención del título, del Reglamento para la Admisión, Permanencia y Egreso de los Alumnos de la Universidad Autónoma de Nuevo León, aprobado el 8 de agosto de 2019 y en el Reglamento Interno de la Escuela o Facultad.

Se procedió a tomar la Protesta de Ley por el Presidente del Comité de Titulación y en cumplimiento de lo dispuesto por los preceptos legales y reglamentarios, firman la presente acta los profesores(as), ante la presencia del Secretario del Comité que da fe.

PRESIDENTE	SECRETARIO
[illegible]	[illegible]
DR. ROGER ZIRAHUEN RÍOS MERCADO	DRA. MARÍA ANGÉLICA SALAZAR AGUILAR

VOCAL

[illegible]

DR. VINCENT ANDRÉ LIONEL BOYER

El suscrito, Director de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA, CERTIFICA que las firmas que aparecen en la presente acta son auténticas y las mismas que utilizan los C.C. profesores mencionados en ella.

U.A.N.L.
DIRECCIÓN

[illegible]

DR. ARNULFO TREVIÑO CUBERO

El C. Secretario General de la Universidad Autónoma de Nuevo León, CERTIFICA que la firma del C. Director de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA que aparece en la presente acta, es auténtica.

[illegible]

DR. JUAN PAURA GARCÍA

SECRETARÍA GENERAL
DEPARTAMENTO ESCOLAR Y DE ARCHIVO

jmor/desb El recibir de conformidad este documento académico, compromete al interesado a dar el uso legal y correcto del mismo. Quien altere cualquiera de las partes que lo conforman, invalida inmediatamente su autenticidad, haciéndose acreedor a la aplicación de las sanciones previstas en las leyes y reglamentos de la UANL; independientemente de los efectos legales que procedan. IA 24/06/24

UANL-T
013744

ORIGINAL

LPT-23

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

INGENIERÍA EN TECNOLOGÍA DE SOFTWARE

.

Junio 2024

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions

por

Eduardo Salazar Treviño

como requisito parcial para obtener el grado de

ingeniería en tecnología de software

.

Junio 2024

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Los miembros del Comité de Tesis recomendamos que la Tesis "A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions", realizada por el alumno Eduardo Salazar Treviño, con número de matrícula 1847972, sea aceptada para su defensa como requisito para obtener el grado de Ingeniería en Tecnología de Software.

El Comité de Tesis

Dr. Roger Z. Ríos Mercado
Director

Dra. María Angélica Salazar Aguilar
Revisor

Dr. Vincent André Lionel Boyer
Revisor

Vo. Bo.

Dr. Fernando Banda Muñoz
Subdirector Académico

San Nicolás de los Garza, Nuevo León, Junio 2024

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Los miembros del Comité de Tesis recomendamos que la Tesis "A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions", realizada por el alumno Eduardo Salazar Treviño, con número de matrícula 1847972, sea aceptada para su defensa como requisito para obtener el grado de Ingeniería en Tecnología de Software.

El Comité de Tesis

[illegible]

Dr. Roger Z. Ríos Mercado
Director

[illegible]

Dra. María Angélica Salazar Aguilar
Revisor

[illegible]

Dr. Vincent André Lionel Boyer
Revisor

Vo. Bo.

[illegible]

Dr. Fernando Banda Muñoz
Subdirector Académico

San Nicolás de los Garza, Nuevo León, Junio 2024

A los gigantes

en cuyos hombros estoy parado.

Contents

List of Figures

List of Tables

Agradecimientos

Agradezco a mis señores padres, Eduardo Salazar Naranjo y Laura Marcela Treviño Lara, por haberme apoyado en cumplir todos mis sueños y metas. Ahora que culmino mi primera etapa de estudios profesionales quiero agradecerles todo lo que han hecho por mí hasta llegar a este punto, se los debo todo a ustedes, las palabras no son suficientes para darles las gracias por su sacrificio y esfuerzo durante toda mi vida, que me han permitido soñar y cumplir esos sueños. Agradezco también a mis hermanas, Carla Elizabeth y María José por su apoyo moral y amistad fraternal, a lo largo de toda mi vida y en especial en mi carrera.

Quiero agradecer a mi principal asesor, director de tesis y mentor, el Doctor Roger Z. Ríos Mercado por toda su guía brindada en este proceso. Ha sido un camino muy largo el que he recorrido siendo impulsado y guiado por el. No existen palabras para expresar la gratitud que siento con y hacia el por su infinita paciencia, sabiduría, comentarios y retroalimentación de estos dos años. Nada de esto sería posible sin él. Adicionalmente quiero agradecer profundamente también a la Doctora Diana Lucía Huerta Muñoz, también mi coasesora por todos sus comentarios y ánimos en el proceso, si algún día sentía que iban mal las cosas, la doctora. siempre me podía animar con su optimismo, sus comentarios y atención al detalle han sido invaluables en este proceso.

Gracias a mis revisores, la doctora María Angélica Salazar Aguilar y al doctor Vincent André Lionel Boyer por su tiempo y esfuerzo en la revisión y mejora de este trabajo.

Agradezco a la Facultad de Ingeniería Mecánica y Eléctrica el haberme permitido ser alumno de la mejor institución ingenieril de México, a la Universidad Autónoma de Nuevo León por acogerme como alumno desde el bachillerato y brindarme las herramientas y oportunidades que hoy estoy aprovechando. También agradezco al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) por su apoyo económico con la beca de Ayudante de Investigador del SNI nivel III y a la clase trabajadora de México, con cuyos impuestos recaudados se financían estos programas.

Asimismo a pesar de no formar parte oficialmente del Programa del Posgrado en Ingeniería de Sistemas (PISIS) en la FIME, quiero agradecer también a sus maestros y alumnos por también haber colaborado en este proceso, la doctora Elisa, la doctora Sara Elena, el maestro Said y el doctor Sergio.

Por último, gracias a todos los amigos que he hecho en el camino: Neto, Pablo, Chava, Saúl, Emmanuel, Isaac, Uriel, Betty, Tali y Rodolfo.

Resumen

Eduardo Salazar Treviño.

Candidato para obtener el grado de Ingeniería en Tecnología de Software.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Número de páginas: 135.

Objetivos y método de estudio: Analizar, diseñar e implementar algoritmos heurísticos eficientes que proporcionen soluciones de buena calidad para un problema de optimización combinatoria de diseño territorial de una institución microfinanciera, así como comparar los resultados y tiempos de ejecución de los algoritmos propuestos con respecto a los resultados obtenidos por software de optimización exacta.

El problema de diseño territorial se refiere a la partición de un conjunto B de unidades geográficas básicas y un conjunto S de posibles centros territoriales en p territorios, respetando restricciones espaciales y de planificación.

Contribuciones y conclusiones: En esta tesis se ha desarrollado e implementado una metaheurística basada en un Procedimiento de Búsqueda Adaptativa Ávida Aleatorizada (*GRASP* en inglés) para abordar el problema de estudio, demostrando eficacia al obtener soluciones comparables a las mejores soluciones encontradas por software de optimización exacta y con mejor tiempo de cómputo.

Firma del director: ______________________
Dr. Roger Z. Ríos Mercado

Abstract

Eduardo Salazar Treviño.

As a candidate to obtain a degree in Software Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of the study: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Number of pages: 135.

Objectives and Study Methods: The purpose of this thesis is to analyze, design, and implement efficient heuristic algorithms that provide good quality solutions for a combinatorial optimization problem arising in the territorial design for a microfinance institution.

The territorial design problem refers to the partitioning of a set B of basic geographical units and a set S of possible territorial centers into p territories. This is done while meeting spatial and planning constraints.

Contributions and Conclusions: A Greedy Randomized Adaptive Search Procedure (GRASP) has been developed and implemented to address the problem under study, demonstrating efficiency in obtaining solutions comparable to the best solutions found by exact optimization algorithms, with shorter running times.

Director signature: ____________________________
Dr. Roger Z. Ríos Mercado

Resumen

Eduardo Salazar Treviño.

Candidato para obtener el grado de Ingeniería en Tecnología de Software.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Número de páginas: 114.

Objetivos y método de estudio: Analizar, diseñar, e implementar algoritmos heurísticos eficientes que proporcionen soluciones de buena calidad para un problema de optimización combinatoria de diseño territorial de una institución microfinanciera, así como comparar los resultados y tiempos de ejecución de los algoritmos propuestos con respecto a los resultados obtenidos por software de optimización exacta.

El problema de diseño territorial se refiere a la partición de un conjunto B de unidades geográficas básicas y un conjunto S de posibles centros territoriales en p territorios, respetando restricciones espaciales y de planificación.

Contribuciones y conclusiones: En esta tesis se ha desarrollado e implementado una metaheurística basada en un Procedimiento de Búsqueda Adaptativa Ávida Aleatorizada (*GRASP* en inglés) para abordar el problema de estudio, demostrando eficacia al obtener soluciones comparables a las mejores soluciones encontradas por software de optimización exacta y con mejor tiempo de cómputo.

Firma del director: [illegible]
Dr. Roger Z. Ríos Mercado

Abstract

Eduardo Salazar Treviño.

As a candidate to obtain a degree in Software Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of the study: A Greedy Randomized Adaptive Search Procedure for a Territory Design Problem Arising in Microfinancial Institutions.

Number of pages: 114.

Objectives and Study Methods: The purpose of this thesis is to analyze, design, and implement efficient heuristic algorithms that provide good quality solutions for a combinatorial optimization problem arising in the territorial design for a microfinance institution, as well as comparing the results and running times of the proposed algorithms with respect to the results obtained by exact optimization software.

The territorial design problem refers to the partitioning of a set B of basic geographical units and a set S of possible territorial centers into p territories. This is done while meeting spatial and planning constraints.

Contributions and Conclusions: A Greedy Randomized Greedy Adaptive Search Procedure (*GRASP*) has been developed and implemented to address the problem under study, demonstrating efficiency in obtaining solutions comparable to the best solutions found by exact optimization software, with shorter run-times.

Director signature: [illegible]
Dr. Roger Z. Ríos Mercado

Chapter 1

Introduction

A classic Territorial Design Problem (TDP), also known as a Districting Problem, organizes and partitions a given set B of small geographical units or *Basic Units* (BUs), and a set S of territory centers into p larger geographical clusters called territories or districts according to spatial constraints such as compactness and contiguity, and planning constraints in order to balance the magnitude of the performance of several activities across all territories, such as their economic performance (sales, total workload, number of customers served) or specific physical needs of each center (Ríos-Mercado and López-Pérez, 2013).

A district is called *contiguous* if it is possible to travel between each pair of basic units of the district without leaving the district. *Compactness* refers to round-shaped, undistorted territories without holes. The reason for preferring compact districts is the need of maintaining contiguity, as it minimizes daily travel distances within the territories. In terms of compactness, a dispersion measure based on the objective function of several classic location problems, such as the p-center problem or the p-median problem, is usually considered (Laporte, Nickel, and Gama, 2019). Since p is not equal to $|S|$, the first decision to be taken is which of all the S possible centers must be used to serve the BUs. Once the p centers are known, the individual allocation of BUs to their center can be decided.

1.1 Motivation

The purpose of this study is to provide a generalization for microfinancial institutions that wish to design territories representing where to open a service branch and which clients will be served by that branch. Risk balancing among the branches is extremely important for this type of business, as a failure in a single branch node can compromise the entire network.

The model described in this thesis is motivated by a previous model described in López, Ekin, Mediavilla, and Jimenez (2020). In the present model, risk balancing is modeled as a constraint, whereas in the original model it is a part of the objective function. Moreover, the model from López et al. (2020) was applied to an existing territorial design, so the objective function also included a term related to retaining as much as possible the same territorial features of the original design. Additionally, in our model we assume we do not have connectivity constraints. Finally, the previous model is solved using its mathematical formulation as a Mixed Integer Linear Programming problem.

The proposed model can be viewed as a p-median problem with multiple capacity constraints. Given that even the uncapacitated p-median problem is $\mathcal{NP}$-hard (Ríos-Mercado and Escalante, 2016), our TDP is also $\mathcal{NP}$-hard. The $\mathcal{NP}$-hard nature of the problem is what motivates us to pursue the use of heuristic approaches for solving it in a reasonable time.

1.2 Objectives

- Studying and analyzing a generalization of a territorial design problem with activity measures, risk, and center types balancing constraints.
- Designing and implementing efficient and robust heuristic and metaheuristic

algorithms for the problem.

- Demonstrating the effectiveness of solving such a problem using the proposed heuristic framework.

- Comparing the performance of the proposed robust metaheuristic with respect to a state-of-the-art exact solver.

1.3 Contribution

In this work, a Greedy Randomized Adaptive Search Procedure metaheuristic is proposed to efficiently solve a territorial design problem of a microfinancial institution.

The key components of this metaheuristic are: a robust constructive heuristic which guarantees near-feasible solutions, repaired later by a local search phase. Once a solution is feasible, the local search changes its priority towards improving the objective function using several local neighborhood structures considering a cyclical neighborhood exploration strategy. The proposed metaheuristic framework which involves parallel computing has been assessed and evaluated over a wide range of test instances.

By comparing the results of the proposed metaheuristic versus the solutions obtained from the mathematical model solved by a general-purpose optimization solver, we observe that the heuristic outperforms off-the-shelf optimizers in terms of total computing time and solution quality is within an acceptable relative optimality gap to the best bound found by the solver.

This thesis is organized as follows: In Chapter 2, we provide the literature review related to the problem. In Chapter 3, the definition of the problem and the mathematical formulation are presented. In Chapter 4, the proposed algorithms are described. Chapter 5 presents the computational experience where the proposed

metaheuristic is fully assessed. Finally, in Chapter 6, we provide the conclusions for this research work.

Chapter 2

Literature Review

In this chapter, specifically in Section 2.1, we analyze the literature for the Territorial Design Problem (TDP) and an overview on microfinancial institutions and their importance. Sections 2.2 and 2.3 describe the problem and provide the representation of feasible solutions. In Section 2.4, we will model the problem based on the previous description.

This TDP identifies potential branch offices (territory centers) and allocates clients to their respective office, balancing several constraints while minimizing the distance of the clients and their assigned institution branch.

The territory centers must be distributed across different hosts' business lines. Due to the *micro* nature of the institution, there are not enough resources to build new physical offices to give service, instead, those offices must be opened in existing facilities of different businesses and services, such as restaurants, gas stations, pharmacies, etcetera, to serve clients alongside the main activity performed by the establishment. Therefore, all centers must be distributed evenly across the different hosts' business lines.

The aim is to determine what centers to open from a set of possible locations, such that the sum of distances from the basic units to their assigned institution branch is minimized, subject to balance criteria: the required workload for each

BU, total quantity of money involved in the loans for each BU, and total utilities generated for each BU. These criteria must be balanced around a target value, and since the exact value will not be met perfectly, a tolerance parameter is introduced to allow some leeway. Finally, the sum of the risk involved in each loan for every center must not exceed a given threshold.

2.1 Districting

Districting problems (DP) or territorial design problems arise in many different contexts; from classical areas such as political districting, sales alignment, or public service, to more recent applications in police districting, waste management, commercial or healthcare (Ríos-Mercado, 2020).

The challenge of establishing political boundaries can be perceived as partitioning an administrative region, like a city or state, into subdivisions for electing political representatives, often known as *political districting.* This issue holds significant importance in democratic systems where each district selects representatives for a legislative body. To avoid politicization of the redistricting process, numerous states have established legislative and practical standards. Several criteria are used to make the districting decisions: demographic criteria such as population, minority representation and voter equality; geographic criteria that model the previously discussed compactness and contiguity features and, finally, political data.

The crucial but arduous work of developing sales and services territories is shared by all businesses that employ a sales team and require the market area to be divided into zones of accountability. Similarly, the issue of defining service territories for customer or technical facility support is closely related. In these cases, similar standards are typically used to create regions for personnel of service. There are multiple reasons for adjusting current territories, or for planning new territories. First, any increase or decrease in the number of sales or service personnel undoubtedly

requires adjustments to the territories. Additional justifications include enhancing the current workforce's reach or to distribute workload uniformly among them. Furthermore, shifts in customer demographics or the roll-out of new products require a restructuring of boundaries.

The sales territorial design problems are closely related to the problem addressed in this work. Several criteria are overlapped between both of them, for instance, the number of territories is predetermined and the presence of basic units, that represent individual customers assigned to larger geographical units with exclusive assignments are considered. Sales districting problems also have activity-related criteria that represent the performance of the products or service that an organization may offer. Dealing with sales and service territory districting issues, there is often a pursuit for districts that balance one or more characteristics (referred to as *activity measures*). This standard shows the relationship between territories to ensure equitable treatment of all sales personnel (Kalcsics, 2005).

The creation of service districts emerges in multiple scenarios. One of them is focused on social infrastructures: hospitals or public utilities. There are cases where districts are required to assign each resident to a specific facility for service provision, such as routine health check-ups, or to delimit areas of responsibility for home-care visits by healthcare staff such as nurses or physiotherapists. The objective is to identify joint districts that are easily reachable via public transport and ensure a balanced workload based on service and travel time or aim for optimal capacity usage of the social facility.

Despite all the different criteria of all the possible applications of districting, there is no definitive mathematical model to represent each problem. Depending on each context, the formulation of the problem and the solution approach are different. The compactness criterion is also something that depends on the specific problem as there is no concrete definition of compactness since it depends on the representation of the basic units. In political districting, compactness is preferred to be represented

as specific territorial shapes, while in sales and services districting, distance-based compactness is preferred.

For comprehensive surveys and discussion on districting models, algorithms, and applications, the reader is referred to the works by Kalcsics and Ríos-Mercado (2019) and Ríos-Mercado (2020).

The studied TDP considers a distance-based dispersion measure, which can be represented in several manners, resulting in several models and solutions. The commonly used dispersion measures are those used for the p-Median Problem (pMP) and the p-Center Problem (pCP). In the pMP, the objective is to place p facilities (or medians) in such a way that the total weighted distance (or cost) between demand points and their assigned facilities is minimized (Laporte, Nickel, and Gama, 2019). This is particularly suitable for situations where the total travel (or service) cost needs to be minimized, for example: to determine the optimal locations for warehouses to minimize the total transportation cost. In contrast, the pCP aims to locate p facilities (or centers) such that the maximum distance from a demand point to the nearest facility is minimized (Medrano, 2020). This essentially minimizes the worst-case scenario in terms of the distance traveled by a customer to reach a facility. This might be used, for example, to locate emergency facilities such as fire stations or hospitals, where the aim is to minimize the longest distance that must be traveled in an emergency. In the context of districting, Salazar-Aguilar et al. (2011) present a computational study of both pMP and pCP objectives used within a territory design framework. For this thesis, the dispersion is measured using a function from the p-Median problem.

2.2 Microfinancial institutions

Microfinancial institutions are important to reduce income inequality and poverty (Khandker, 2005). They offer credit and other financial products to sectors of the

population that are normally rejected by commercial banks as they represent high-risk with low profit options. They have risen as an alternative to regular banking, especially in countries with developing economies (Bruton, Khavul, Siegel, and Wright, 2015). Due to their fragile nature, the risk associated with the loans granted must be balanced across the branch offices.

Compared to traditional banking systems, the MFIs do not operate nor own branch offices, instead, only overseeing credit operations. These branch offices or subsidiaries operate as part of local businesses (gas stations or grocery stores) that allow MFIs to reduce operating and construction costs. MFIs face risks that normal banks do not. Traditional banking systems discourage the practice of lending money to people who do not have guarantees to secure the loans and are unable to pay them back, excluding vulnerable sectors of the population.

A relationship of mutual rewards with the branches is established so that the network remains sustainable in the long term. Balancing risks between different branches is important, since there is no universal definition of risk, measures can be based on statistical deviations or problem-specific functions.

In this thesis, a function based on the problem's variance is used, which will be referred to as the variance of excessive gain. For each BU, a gain variance corresponds to a higher risk in which small establishments are not exposed to high variance. Furthermore, each open branch office and their corresponding served clients, forms a territory or district, with the office being the territory center and the clients the basic units.

Also, each type of branch must manage different levels of risk in order to control and balance the tolerance of each territorial center. For this, the gain variance for each BU is first estimated using historical customer data. Then, the total gain variance within the centers is used as an internal risk measure to obtain information about the customer retention rate. This rate estimates how probable it is that a client continues to be such. Therefore, the MFI pre-determines an upper limit for

the total weighted gain variance, for each territorial center. Subsequently, customers should be assigned to territories in such a way that the total variance of excessive gain is minimized. Deviations towards lower limits are not taken into account. A more detailed description of the studied problem in this thesis is provided below.

2.3 Districting applied to MFIs

Territorial design models are rarely applied to traditional banking in the literature. The most important contribution to the literature can be found in Monteiro (2005) and Monteiro and Fontes (2006), in which traditional MILP formulations are applied to locating and maintaining bank-branches, closing or opening existing branches by posing a coverage problem. However, these contributions recognize the shortcomings of the MILP approach and introduce local search heuristics that start from a partial solution provided by an optimization solver. These contributions take into consideration several costs of operation, location, as well as sizing, thus they are not exactly districting formulations.

Districting, specifically applied to MFIs, first appeared in López, Ekin, Mediavilla, and Jimenez (2015) where a two phased mixed integer program, hybridized with several heuristics, is used to solve a real-life instance from a MFI located in Monterrey, México. This model presents the risk balancing as a parameter in the objective function as well as introducing a similarity parameter, which motivates the new design to retain the features of a previous design. Additionally, another constraint limits the maximum amount of distance traveled by the customers. Some BUs are constrained to be allocated to different territories, introducing disjoint assignment constraints. Finally, contiguity constraints, which evaluate that every district induces a connected graph between all the BUs and the corresponding territory center, are included. This type of constraint in specific grows exponentially, so solving medium to large instances becomes intractable in a reasonable computing time. By using heuristics in several phases, they were able to reduce their problem complexity

in order to use an exact solver.

A follow-up work was published in López, Ekin, Mediavilla, and Jimenez (2020), in which the heuristics used to set some variables for the solver were slightly improved on, using Geographic Information System (GIS) software to provide interpretability to the MFI and properly implement the solution found. As it can be seen, there have been no previous attempts to apply districting to MFIs in a broader context, not tied to a specific case study.

In this thesis, we propose a generalization of the model of López, Ekin, Mediavilla, and Jimenez (2020) so it can be applied in more cases by removing: the hard constraints on maximum distance, the disjoint assignments constraints, and the contiguity constraints, as well as simplifying the objective function by considering the risk term as a constraint and removing the similarity to a previous plan parameter. Notice that most territorial design problems have a district center c_k for each district D_k that coincides with a basic unit, i.e. $c_k \in B$: however, in this thesis, the district center is a member of another set, $c_k \in S$. By reformulating the problem and solving it with an exact solver, we can verify that the computation is not trivial and an optimal solution cannot be reached in reasonable time; thus, we encourage the use of heuristics and metaheuristics to solve the problem. This thesis is the first work, to the best of our knowledge, to propose a generalized heuristic framework for districting applied to MFIs and, subsequently, the first to propose metaheuristics.

Chapter 3

Problem description and Mathematical model

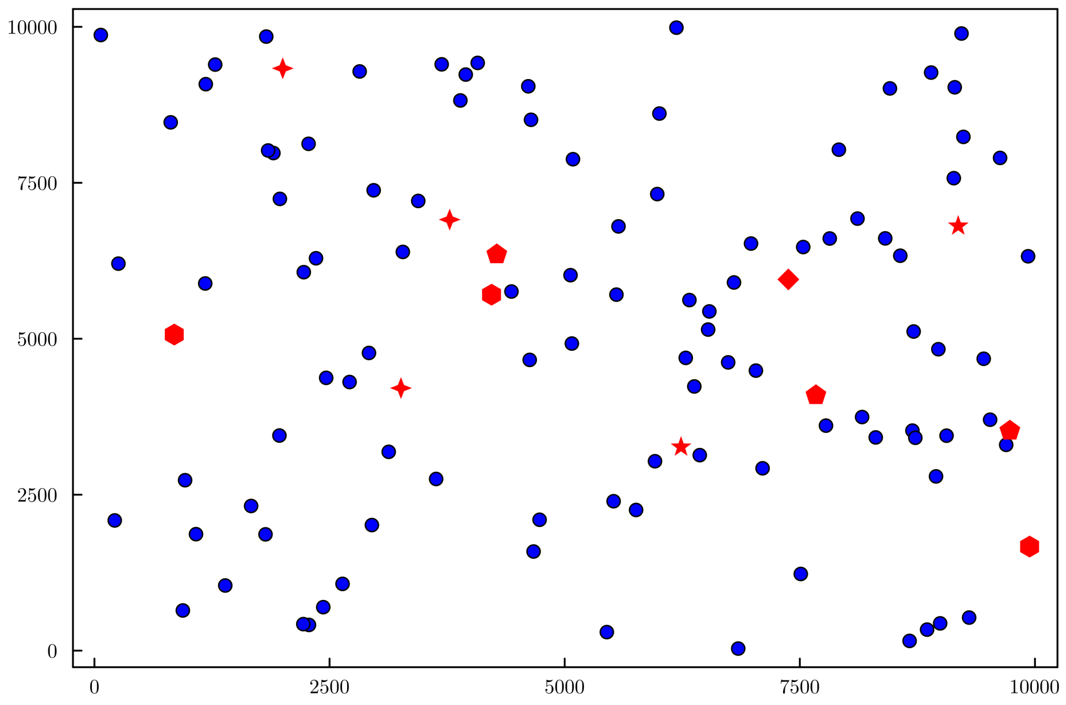
3.1 Problem description

Given a set S of possible territory centers, a set B of BUs, a distance matrix d and a p number of centers to be open, let the set M of activities to evaluate and their corresponding activity values v_m^j, where $m \in M$ and $j \in B$, the sum of these values for all the BUs assigned to a territory center must be in a range defined by multiplying the target measure μ_m^i for each activity $m \in M$, $i \in S$, and a t_m tolerance parameter. Additionally, given each individual risk value r_j for $j \in B$, the sum of all the risk values of the assigned BUs to a center i must not exceed a risk threshold β_i.

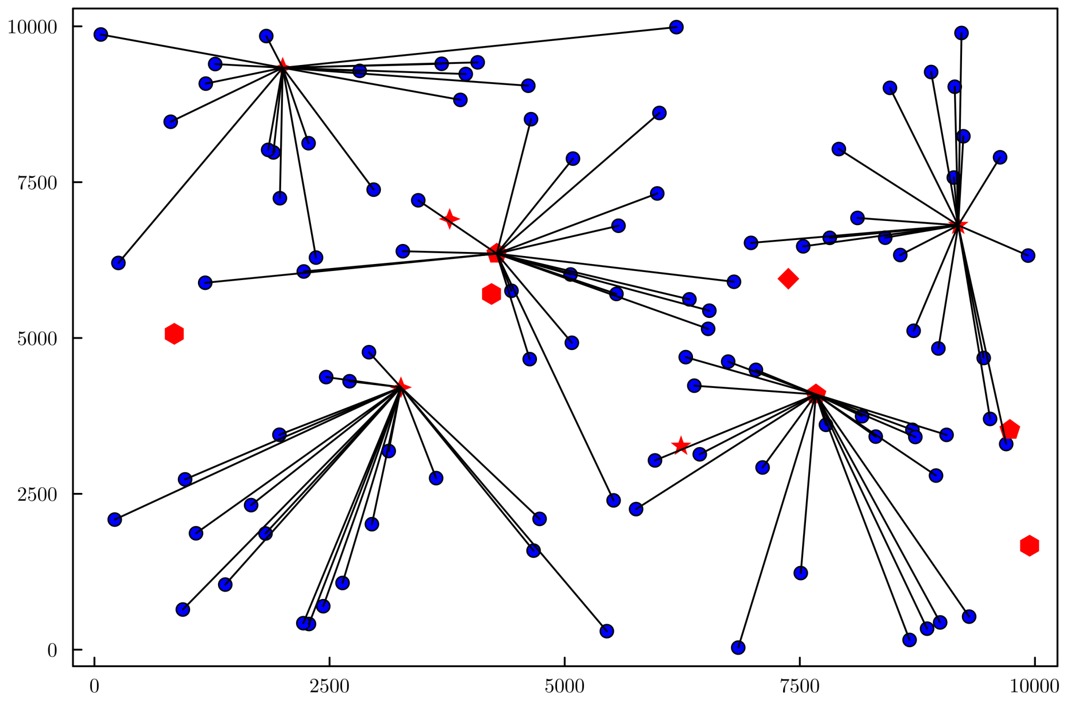
Given a set K of territory types, an associated T_{ik} value is assigned for each center $i \in S$ of type k, for $k \in K$. A lower bound L_k and an upper bound U_k, are determined for the number of centers used for each type, $\forall k \in K$. Finally, each BU $j \in B$ must be assigned only once. The territorial design problem aims to find the location of the p centers to be opened and the allocations of all BUs to their corresponding centers such that the sum of the distances in d between each BU and its assigned center is minimized, taking into account the previous constraints.

3.2 Illustrative example of feasibility

Figure 3.1a shows a basic example of an instance of the problem with BUs and territory centers, randomly located. The BUs are represented as blue circles, and the different red geometric figures are used to represent the different center types, with $K = 5$. In contrast, Figure 3.1b illustrates the optimal solution to the instance, minimizing the total distance between each BU and their assigned territory center. It is important to emphasize that there are only 5 centers available to serve BUs, as $p = 5$. In an instance of larger size, the optimal solution that minimizes the sum of distances may allocate individual BUs to centers that are not their nearest, in a counter-intuitive fashion. These allocations arise as the territories are constrained by the activity measures and the risk thresholds, as seen in Figure 3.2b.

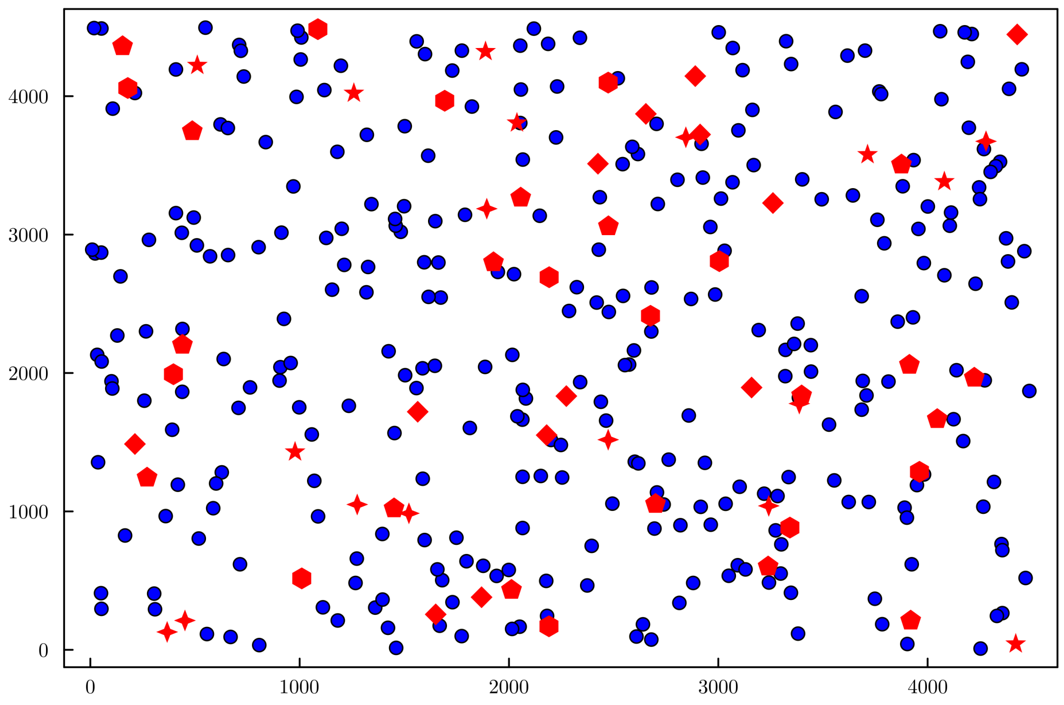


(a) Example of an instance with $|B| = 100$, $|S| = 12$, $p = 5$

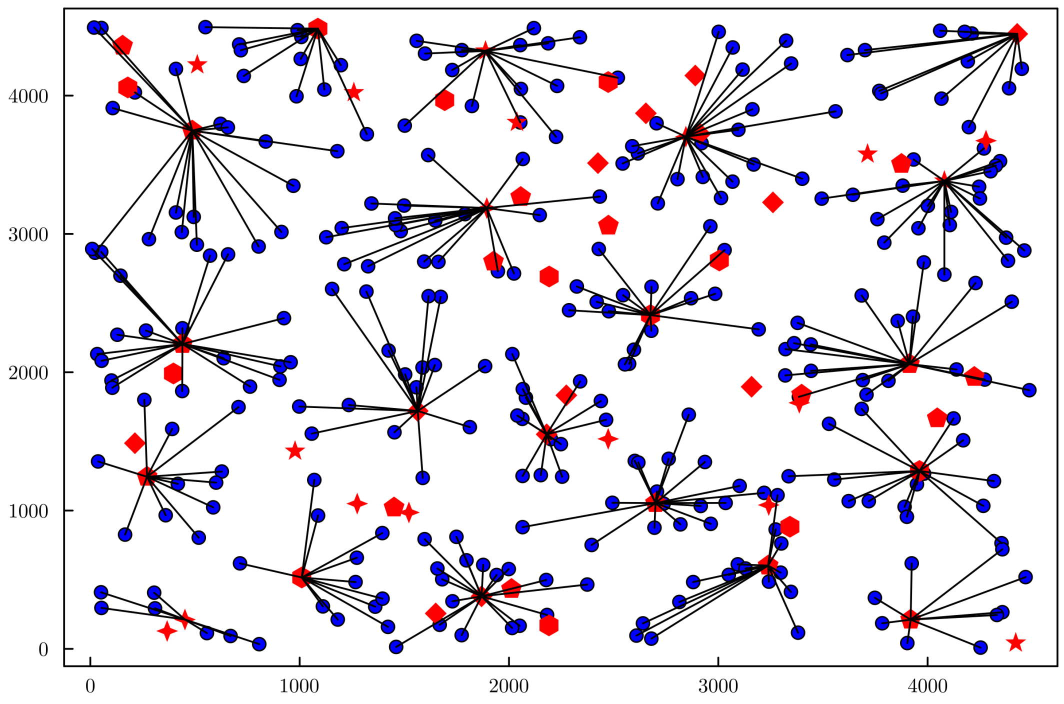


(b) The optimal solution for the instance.

Figure 3.1: Optimal solution with a weight of 208567 for a small size instance



(a) Instance with $|B| = 300$, $|S| = 60$, $p = 20$



(b) Optimal solution for the above instance.

Figure 3.2: Optimal solution for a larger instance

3.3 Mathematical model

In this section, the mathematical model used in this thesis is described, studied, and analyzed. This model is based on the formulation presented by López, Ekin, Mediavilla, and Jimenez (2020) with the following modifications. The objective function in the original model has three terms to minimize: the sum of distances, the risk of the allocations, and a similarity factor that measures how similar the new territorial design is to an existing design. As this version of the problem models the risk as a constraint and has no previous territorial design, the objective function only minimizes the distance. Additionally, the previous model has a specific connectivity constraint set that requires all territories to induce a connected graph with the BUs and their corresponding center. This set has an exponential size, thus they limit this restriction to a given set of territory centers only. On the other hand, there is also a restriction that ensures that certain pairs of BUs are allocated to different territories due to geographical or management requirements. Both of the connectivity and the exclusive assignments are not present in the current model given the additional complexity they would represent.

- Sets and parameters:

- S : Set of possible territory centers
- B : Set of possible BUs
- M: Set of activities to evaluate
- K: Set of territory center types
- d_{ij}: Distance between center i and BU j, $i \in S, j \in B$
- μ_m^i: Target of activity m measured at center i, $i \in S$, $\forall m \in M$
- v_m^j: Measure of activity m at BU j, $j \in B$, $\forall m \in M$

- t_m: Tolerance of activity m measure, $\forall m \in M$
- r_j: Risk value at BU j, $j \in B$
- β_i: Risk threshold at center i, $i \in S$
- $T_{ik} = 1$ if center i is of type k, 0 otherwise, $i \in S, k \in K$
- $L_k, U_k =$ Lower and upper bounds of number of centers of type k to be located, $k \in K$
- $p =$ Number of centers to be used

- Decision variables:

- $Y_i = 1$ if center i is open; 0 otherwise.
- $X_{ij} = 1$ if BU j is assigned to center i; 0 otherwise.

Then, the territorial design problem for the MFI is defined as:

$$\min \sum_{i \in S, j \in B} d_{ij} X_{ij} \tag{3.1}$$

Subject to

$$\sum_{i \in S} X_{ij} = 1, \qquad \forall j \in B \tag{3.2}$$

$$X_{ij} \leq Y_i, \qquad \forall i \in S, j \in B \tag{3.3}$$

$$\mu_m^i (1 - t_m) Y_i \leq \sum_{j \in B} v_m^j X_{ij} \leq \mu_m^i (1 + t_m) Y_i, \qquad \forall i \in S, \forall m \in M \tag{3.4}$$

$$L_k \leq \sum_{i \in S} T_{ik} Y_i \leq U_k, \qquad \forall k \in K \tag{3.5}$$

$$\sum_{i \in S} Y_i = p \tag{3.6}$$

$$\sum_{j\in B} r_j X_{ij} \leq \beta_i, \qquad \forall i \in S \tag{3.7}$$

$$X_{ij} \in \{0,1\}, Y_i \in \{0,1\}, \qquad \forall i \in S, \forall j \in B \tag{3.8}$$

The objective function (3.1) minimizes the sum of the distances between the BUs and their assigned center. Constraints (3.2) state that each BU j must be assigned to a single center i. Constraints (3.3) check that the BUs are only assigned to open centers. Constraints (3.4) establish that the activity m measured for each territory i, must be within a tolerance range defined by the t parameter. Notice that because of the discrete structure of the problem it is impossible to perfectly balance each territory. Constraints (3.5) limit the located centers' types within their corresponding lower and upper bound. Constraint (3.6) ensures that only p territory centers are considered. Constraints (3.7) specify that the total risk value for each territory, must not surpass its defined risk threshold. Finally, the binary nature of the variables is expressed in Constraints (3.8).

Therefore, for an instance with $|S|$ possible centers and $|B|$ basic units, the total number of decision variables is $|S||B| + |B|$.

The problem is $\mathcal{NP}$-hard. This can be argued as follows. Clearly, the problem is in $\mathcal{NP}$ because one can check feasibility of a given solution in polynomial time. Now, by reduction, if we take a special case of our problem with a very large value for parameters t_m, U_k, and β_i and a value of zero for all L_k, Constraints (3.4), (3.5), and (3.7) become redundant so we are left with a p-median problem. Thus, we have proven that the p-median problem is polynomially reducible to our problem, and since it is well-known that the p-median problem is $\mathcal{NP}$-hard, so is our problem.

Chapter 4

A Greedy Randomized Adaptive Search Procedure

In this chapter, a Greedy Randomized Adaptive Search Procedure metaheuristic is presented, which includes several well-known heuristics adapted to the problem studied in this thesis.

Although some exact optimization approaches exist for territorial design problems (Salazar-Aguilar, Ríos-Mercado, and Cabrera-Ríos (2011), Ríos-Mercado and Bard (2019), Sandoval, Díaz, and Ríos-Mercado (2020)), the vast majority of the work has been on metaheuristics, given the inherent complexity of territory design problems.

First introduced in Feo and Resende (1989) as a "probabilistic heuristic", the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic is a multi-phase iterative process designed for solving combinatorial optimization problems. It combines elements of greediness and randomness to effectively navigate the solution space. The two main components of GRASP are the construction phase and the local search phase, which work to iteratively improve upon solutions.

In the construction phase, an initial feasible solution is built, starting from an empty solution and iteratively adding elements to it. At each step, a candidate

list of possible next elements is generated based on a greedy function. The key aspect of GRASP's construction phase is the incorporation of randomness: instead of choosing the best candidate (as in a purely greedy approach), one element is randomly selected from a subset of potential candidates. This subset, often referred to as the Restricted Candidate List (RCL), is formed by selecting elements that are within a certain α quality threshold or a fixed size. In this thesis, we use the first approach. The randomness introduced at this stage helps in diversifying the search, avoiding premature convergence to local optima.

Once the construction phase generates an initial solution, the local search phase seeks to improve it through an iterative process. The solution is refined by exploring its neighborhood, which is a set of solutions that can be reached through small modifications (like swapping, adding, or removing elements). The local search continues until a local optimum is found, meaning no better solution can be found in the neighborhood. This process is repeated until a stopping criterion is met, typically, a fixed number of iterations.

Since each iteration is independent, generating a different initial solution every time the construction phase is performed, which is then improved during the local search phase, GRASP has the potential to be parallelized, running each iteration on a different computer core thread. This repeated process allows the algorithm to explore various parts of the solution space, naturally balancing diversification (exploring different regions of the solution space) and intensification (searching a promising region). The randomness in the construction phase promotes diversification, while the local search phase focuses on intensification (Feo and Resende, 1995).

GRASP has been very successfully applied to a class of districting and territory design problems (Ríos-Mercado and Fernández, 2009; Fernández et al., 2010; Ríos-Mercado and Escalante, 2016) including multi-objective optimization formulations (Salazar-Aguilar, Ríos-Mercado, and González-Velarde, 2013).

In order to build a GRASP metaheuristic framework, a constructive and local

search heuristics must be developed first, to integrate them into the framework. In this chapter we define what heuristics and strategies were explored to build the proposed GRASP. The pseudocode of the GRASP is presented in Algorithm 1.

Algorithm 1 `GRASP(Instance,` $\alpha_l, \alpha_a, i_{\max}$`)`

Input: Instance = instance of the problem; α_l = RCL threshold parameter for location phase; α_a = RCL threshold parameter for allocation phase; $i_{\max}$ = maximum number of iterations.

Output: X^* allocation matrix of BUs, Y^* location vector of centers, z^* objective function value

```
1:  $X^* \leftarrow \emptyset$
2:  $Y^* \leftarrow \emptyset$
3:  $z^* \leftarrow \infty$
4:  $i \leftarrow 0$
5:  $z \leftarrow f(X, Y)$
6:  while $i < i_{\max}$ do
7:      $Y \leftarrow$ p-disp-grasp(Instance, $\alpha_l$)
8:      $X \leftarrow$ opp-cost-queue-grasp(Instance, $Y, \alpha_a$)
9:      $(X', Y') \leftarrow$ local-search(Instance, $X$, $Y$)
10:     if $f(X', Y') < z^*$ then
11:         $X^* \leftarrow X'$
12:         $Y^* \leftarrow Y'$
13:         $z^* \leftarrow z(X', Y')$
14:     end if
15:     $i \leftarrow i + 1$
16: end while
17: return $X^*, Y^*, z^*$
```

4.1 Constructive Heuristic

The foundational mathematical programming model for districting was introduced by Hess, Weaver, Siegfeldt, Whelan, and Zitlau (1965). In this paper, the *location-allocation* heuristic was first introduced. This heuristic approach breaks down the intertwined tasks of location and allocation inherent to the districting problem into two separate stages. These two stages, one focusing on the selection of territory centers and the other on the assignment of basic units to these centers, are executed in a repetitive cycle until a satisfactory outcome is achieved. The process begins with the location phase, where the centers of future territories are identified, followed by the allocation phase, which involves the distribution of basic units to the selected centers.

The constructive heuristic presented in this thesis follows the same location allocation strategy. In the location phase, the centers, or branches of the MFI, are selected to serve the BUs, while in the allocation phase, the BUs are individually allocated to an open center. Several approaches for both phases were tested and will be presented in the following sections.

4.1.1 Location Phase

Previous studies including Ríos-Mercado, Álvarez Socarrás, Castrillón, and López-Locés (2021) use the location-allocation heuristic successfully by first solving the location phase using p-partition-based heuristics to initialize the set of centroids. Another method for obtaining this initial set of territory centers is by solving the Mixed Integer Linear Programming (MILP) formulation until a feasible solution is found, however, due to the computationally complex nature of the MILP, it does not scale well for larger instances for the problem, this is shown in the next chapter. We believe it is important to once again remind the reader that classic formulations

of the TDP typically contain a single set that includes both the possible territory centers and the basic units, making them interchangeable; however, in this thesis, there are two different sets, one for centers and another one for the BUs. This adds further complexity to our problem as the heuristics used in previous works for this phase need to be adapted to our needs, more specifically the p-dispersion problem heuristic.

Two strategies were explored to use as the location phase algorithm: a p-dispersion problem heuristic and semi-linear programming. Of these, the p-dispersion problem was chosen as the algorithm to use.

4.1.1.1 The p-Dispersion Problem

Due to p being a parameter of the problem, a common strategy is to use p-partition-based heuristics to first divide the set of centers into p subsets, as stated in the previous section. Cano-Belmán et al. (2012) use the p-dispersion problem to locate the centers to be used. The p-dispersion problem, also known as the maxmin problem, aims to select a set of p points that maximizes the minimum distance between any selected pair of points in the set (Erkut and Neuman, 1991).

We are going to use the *Greedy Constructive* (GC) heuristic proposed by Erkut et al. (1994), to try to locate p territory centers from the S set, following the intuitive idea that if they are evenly dispersed, the BUs will have a close enough center to be allocated to. The use of this specific heuristic is reinforced by the contribution of Ravi et al. (1994) that states that there is no polynomial time algorithm that can provide a solution that is not worse by a factor of 2 compared to the optimal solution (unless $\mathcal{P} = \mathcal{NP}$) to the p-dispersion problem, as well as the fact that one of the conclusions of Erkut et al. (1994) is that the selected heuristic presents good enough results for problems in which p is a small proportion of all the points in the problem, which is the case for this TDP. This heuristic is presented in Algorithm 2.

Algorithm 2 GC(D', S, p)

```
Input: $D'$ distance matrix, $S$ set of nodes, $p$ number of nodes to select out of $S$
Output: $Y$ solution set $p$ centers
1: $Y \leftarrow \emptyset$
2: Find $x_1$ and $x_2$ in $S$ such that $d(x_1, x_2) = \max\{D'(x_i, x_j) : 1 \leq i < j \leq p\}$
3: $Y \leftarrow \{x_1, x_2\}$
4: while $|Y| < p$ do
5:     Find $x_i \in S \setminus Y$ such that $d(x_i, Y) = \max\{d(x, Y) : x \in S \setminus Y\}$
6:     $Y \leftarrow Y \cup \{x_i\}$
7: end while
8: return Y
```

The heuristic works by initializing an empty solution set Y on line 1, then it selects the two furthest apart points on the set S as shown on line 2, and adds them to Y.

Then it iterates until the solution set reaches p as its size, from lines 4 to 7. During each iteration, the heuristic finds the node that maximizes the minimum distance to Y and adds it to the solution set, as shown in lines 5 and 6. The distance of an individual node to the set is defined as the minimum distance of the candidate node x_i to each member of Y. Expressed in a more formal notation, the distance between a point x_i and a set Y is defined as $d(x_i, Y) = \min\{D'(x_i, x_j) : x_j \in Y, x_i \neq x_j\}$.

However, one must remember that the territorial centers have an associated T_{ik} individual business type, therefore the types must be also taken into consideration while dispersing the centers, as there is a minimum and a maximum number of centers to be used from each type.

By solving $|K|$ p-dispersion problems, considering only centers of a given $k, k \in K$ type for each problem and using $p = L_k$, we obtain an initial set of selected centers that ensures that fulfills the minimum number allowed for each k. Then, by taking

the union of the resulting centers, the Y vector is obtained as the Algorithm 3 shows. An important thing to note is that the overall p of the problem never equals the sum of all L_k, so a second phase after the union of the k sub-problems is required in order to reach the p centers required to be opened.

This phase solves yet another p-dispersion problem, taking the union of the previous centers as a starting set, and selects the most disperse node to Y, as long as the node respects the U_k constraint for its corresponding k type, until $|Y| = p$.

Algorithm 3 `p-dispersion for Y location`$(d, p, T_{ik}, L_k, U_k, S)$

Input: d distance matrix, p centers to open, T_{ik} binary matrix of center types, L_k, minimum number of centers to be used of type k, U_k maximum centers to be used of type k, S set of centers

Output: Y decision vector of located centers

```
 1: Compute distance matrices, $\forall k \in K$
 2: for $k \in K$ do
 3:     $S_k \leftarrow \{i \in S \mid T_{ik} = 1\}$ { $S_k$ represents all centers of type $k$}
 4:     $P\star \leftarrow$ GC(distance matrix of $k$, $L_k$, $S_k$) {Solve the problem for each $k$, with
        its corresponding nodes and distance matrix}
 5:     $Y \leftarrow Y \cup \{P\star\}$ {Build $Y$ upon the partial solution}
 6: end for
 7: while $|Y| < p$ do
 8:     for node $\in S$ do
 9:         if node $\notin Y$ then
10:             Determine type of node in $S_k$
11:             Check if $U_k$ for this node type is reached; if so, skip this node
12:             Compute the minimum distance from node to all nodes in $Y$
13:             Store this minimum distance.
14:         end if
15:     end for
16:     Select the node with the maximum minimum distance
17:     Add this node to $Y$
18: end while
19: return $Y$
```

A graphical step-by-step example of the algorithm is shown in Section A.6.1.

4.1.1.2 Partial Linear Programming Relaxation

Since our problem is an $\mathcal{NP}$-hard problem according to Karp (1972), large instances are intractable by branch-and-bound methods. By introducing an integer relaxation in the X_{ij} decision variables and leaving the Y_i variables as integers, the idea is to provide to the optimization solver an easier problem to solve, meaning faster times to reach a feasible solution. The solver will provide us with valid results for Y_i, meaning the territory centers have been located. However, this solution may not represent a feasible solution of the original problem; as the X_{ij} variables are not required to be integer. But, we can use the solution values of Y as valid for the proposed heuristic. This approach is referred to as the Partial Linear Programming Relaxation (PLP).

Both of these approaches were used during the experimentation phase of the heuristic to determine which of these two strategies was the one going to be integrated in the GRASP metaheuristic. In the end, the p-dispersion problem heuristic was used as the first phase of the algorithm. The results behind this decision are detailed in Chapter 5.

4.1.2 Allocation Phase

Once the territory centers have been located, the allocation phase of our approach will allocate each individual BU to an open center. This phase is the core of the heuristic as the objective function of the problem directly measures how well clients are allocated to their nearest center whilst complying with the balancing constraints. A possible greedy approach would be to simply allocate all clients to their nearest center, but this provides infeasible solutions. Therefore, a more intelligent decision process must be introduced to achieve the maximum degree of feasibility as possible before passing the constructed solution to a repair phase.

For this phase, two algorithms are proposed, both based on the concept of

opportunity cost. The first algorithm is referred to as the Opportunity Cost with Matrix and the second one is the Opportunity Cost with Queue. The latter was chosen to be used as the allocation algorithm in the GRASP.

4.1.2.1 Opportunity Cost with Matrix

The opportunity cost refers to the value of the next best alternative that is foregone when a decision is made to choose one option over another. It represents the benefits that could have been gained by choosing an specific alternative option. This concept is crucial in economics and decision-making, as it helps in understanding the potential costs involved in not selecting the next best alternative.

Due to the constrained nature of the problem, which is limited in resources, the choice of where to allocate each BU must be made from several mutually exclusive centers. As BUs are assigned to a center, the balancing constraints of both the activities and risk measure will prevent the assignment of any more BUs to that specific center, introducing the scarcity in the mutually exclusive choices available.

By calculating the opportunity cost of all BUs as the difference of the minimal distance available to a center and all other centers, and by storing the results in a matrix with the same size as d, we can now use a heuristic that minimizes the opportunity costs across all BUs, using this matrix as the data structure which will guide the algorithm.

During the allocation phase, we are computing values for a specific i center and a specific j BU. For feasibility checks of the constructed solution, we can keep track and update only the values affected by these specific i and j. By using additional data structures, we introduce the concept of partial evaluation: instead of computing and checking the feasibility of the whole solution, we only update and check it in each individual allocation, making these computations more efficient.

For the algorithm, we start a while loop that iterates until every BU has been assigned. In this loop, we will find the indices of the n largest opportunity costs across the opportunity cost matrix, where n is a parameter of the algorithm. For each index j, which represents the BU, we find the nearest center i to this BU, taking into account that $Y_i = 1$. Afterwards, we compute and check for any violated constraint with the respective allocation of i and j. The number of violated constraints is stored in an array, from which we will pick the allocation that is feasible. If all possible allocations are infeasible, we perform another search across the matrix, checking for the next n nearest centers to the first BU in the list, as well as computing again the constraints, by eventually allocating to the center with the least amount of violated constraints. When the j BU is set to 1 in the X matrix, it is marked as assigned for the algorithm.

The time complexity for the worst case scenario of this allocation strategy is $\mathcal{O}(|B|^2 \cdot |S|)$ time, as there are constant operations of finding either the n maximum or minimum values across the whole matrix, of size $|S| \cdot |B|$.

Notice that the minimum and maximum distances do not really change over the algorithm, thus an initial precomputation of those indices may be useful to speed up times. With those insights, we present the second approach to allocate BUs.

4.1.2.2 Opportunity Cost enhanced with Data Structures

Instead of computing a matrix with all the possible costs, we only compute the opportunity cost from the best possible assignment and the worst one for each BU, storing that difference in a priority queue. A priority queue is a specialized data structure that is similar to a regular queue or stack, but where each element has a priority associated with it. In a priority queue, an element with higher priority is served before an element with lower priority. If two elements have the same priority, they are served according to their order in the queue. In this case, the opportunity

cost is the priority associated with each BU. By traversing this queue, we allocate the BUs in order of largest to least opportunity cost. The n best possible assignments for each BU do not change in distance, therefore we can sort them as well prior to the algorithm, storing the ordered centers in a dictionary. n was set to p to evaluate all the possible assignments.

Algorithm 4 defines the function that calculates such data structures, with *pq* being the priority queue for the opportunity cost and *best_assignments* the dictionary that stores the sorted centers nearest to each client.

Algorithm 4 `compute-structures`(d, Y, B, S, n)

```
Input: Distance matrix $d$, Decision Variable $Y$, $B$ set of BUs, $S$ set of BUs, $n$ top
    assignments
Output: Best assignments dictionary best_assignments, Priority queue pq
 1: Initialize best_assignments as an empty dictionary
 2: Initialize pq as a priority queue with reverse order
 3: for each $j \in B$ do
 4:     Initialize an array costs to store opportunity costs for $j$
 5:     for $i \in Y$ do
 6:         Store $d_{i,j}$ along with index $i$ in costs
 7:     end for
 8:     Sort costs in ascending order of cost
 9:     Store the indices of the top $n$ facilities with the lowest cost in
        best_assignments[$j$]
10:     Calculate the opportunity cost for $j$ as the difference between the highest
        and lowest cost
11:     Add to the queue the $j$ BU and its opportunity cost into pq
12: end for
13: return best_assignments, pq
```

We reuse the data structures to keep track of the constraints and their status

throughout the traversal of the queue. During the queue traversal, we will mark centers as full when their respective activity measures' meet the lower bound target, forcing the algorithm to diversify and choose another center, and ensuring an even distribution among centers. Since the algorithm respects the lower bounds and it simultaneously prevents exceeding upper bounds, feasible solutions are produced more consistently than the opportunity cost with matrix algorithm in a faster time,improving the worst-case scenario from $\mathcal{O}(|B|^2 \cdot |S|)$ to $\mathcal{O}(|B| \cdot |S|)$.

However, it is still possible that a BU is left unassigned due to all of the centers being marked as full, so we must define a function to handle unassigned BUs. This is a very simple function that will simply assign those BUs to the center that has the most capacity left in the β risk threshold.

This algorithm is better in every way than the previous approach of the opportunity cost stored in a matrix, due to the use of data structures and a better understanding of the problem and exploitation of its underlying structure. A graphical step-by-step example of this algorithm is shown in Section A.6.2

4.2 GRASP Metaheuristic Adaptations

The proposed GRASP metaheuristic required an adaptation of the mentioned algorithms to introduce the concept of the Restricted Candidate List (RCL). In this case, we use a value-limited RCL using two α parameters: α_l for the location phase and α_a for the allocation phase. In Chapter 5, we present the results that guide our decision towards using the p-dispersion strategy for the location phase and the opportunity cost enhanced with the queue and other data structures as the allocation phase. In this section, we present the corresponding modifications in order to introduce the RCL in our GRASP.

Algorithm 5 implements a GRASP (Greedy Randomized Adaptive Search Procedure) heuristic for the p-dispersion problem. The procedure is as follows:

Taking as input an instance of the problem comprising a distance matrix d, a set of points S, the number of centers p to select, and a quality threshold α_l for constructing the Restricted Candidate List (RCL), the algorithm identifies the pair $(x_i, x_j) \in S$ that maximizes d_{x_i,x_j}. These points form the initial solution set $Y = x_i, x_j$.

In order to iteratively construct the solution, the algorithm iterates while $|Y| < p$, using the greedy function $\phi : S \setminus Y \to \mathbb{R}$ as $\phi(x_i) = \min\{d_{x_i,x_j} : x_j \in Y\}$, which computes the minimum distance from a candidate point to the current solution set, and determines $\phi_{\max} = \max\{\phi(x_i) : v \in S \setminus Y\}$ and $\phi_{\min} = \min\{\phi(x_i) : x_i \in S \setminus Y\}$. With these two values, it constructs the RCL as
RCL: $\{x_i \in S \setminus Y : \phi(x_i) \geq \phi_{\max} - \alpha_l(\phi_{\max} - \phi_{\min})\}$. This set comprises candidate points that are relatively distant from the current solution set, and finally it randomly selects a point $x_i \in$ RCL and updates $Y = Y \cup \{x_i\}$.

This approach employs a balance between greedy selection and randomization. The greedy component favors points that maximize the minimum distance to the current solution set, while the randomized component introduces diversity by selecting from a set of good candidates. The parameter $\alpha_l \in [0, 1]$ controls the restrictiveness of the RCL: as α_l goes to 0, the algorithm becomes more greedy, whereas larger values of α_l introduce more randomness into the selection process.

It is important to notice that this algorithm is used to solve the k sub-problems for each center type. Once the k solutions have been joined into the larger set, it reuses the same ϕ greedy function, adding centers to the RCL, as long as the candidate center respects the U_k constraint of its corresponding k type.

Algorithm 5 p-`dispersion-grasp`(Instance, α_l)

```
Input: Instance of the problem, $\alpha_l$ quality threshold for the distance
Output: $Y$ solution set of p centers
 1: $d$, $S$, $p \leftarrow$ parameters from the Instance.
 2: $N \leftarrow |S|$
 3: Find $(x_i, x_j)$ such that $d_{x_i,x_j} = \max\{d_{x_i,x_j} : 1 \leq 1 < j \leq |S|\}$
 4: $Y \leftarrow \{x_i, x_j\}$
 5: while $|Y| < p$ do
 6:     Define $\phi(x_i) = \min\{d_{x_i,x_j} : x_j \in Y\}$ for $x_i \in \{1, \dots, N\} \setminus Y$
 7:     $\phi_{\max} \leftarrow \max\{\phi(x_i) : x_i \in \{1, \dots, N\} \setminus Y\}$
 8:     $\phi_{\min} \leftarrow \min\{\phi(x_i) : x_i \in \{1, \dots, N\} \setminus Y\}$
 9:     $RCL \leftarrow \{x_i \in \{1, \dots, N\} \setminus Y : \phi(x_i) \geq \phi_{\max} - \alpha_l(\phi_{\max} - \phi_{\min})\}$
10:     Select random $x_i$ from RCL
11:     $Y \leftarrow Y \cup \{x_i\}$
12: end while
13: return $Y$
```

On the other hand, for the allocation phase, instead of choosing the nearest assignment for the BUs in the priority queue, we define a greedy function $\phi(j) : \min(d_{i,j}, i \in Y)$ that minimizes the distance to a BU j from all available centers in Y, defining $\phi_{\max}$ as the furthest center to j and $\phi_{\min}$ as the nearest center to j. In this case, due to using precomputed data structures and sorting the centers, $\phi_{\max}$ is the last element in the dictionary values for the best assignments of a BU j, and $\phi_{\min}$ is the first element in such values, as they are sorted already. The RCL is constructed as RCL $\leftarrow \{j : \phi(j) \leq \phi_{\min} - \alpha_a(\phi_{\max} - \phi_{\min})\}$. This is detailed in Algorithm 6.

Algorithm 6 `opp-cost-queue-grasp`(Instance, Y, α_a)

Input: Instance of the problem, Y decision variable, α_a: quality threshold for the allocation RCL

Output: X decision matrix of allocations

```
1: Distance matrix $d$, Decision Vector $Y$, $B$ set of BUs, $S$ set of BUs, $p$ centers to be used , $\beta$ risk threshold $\leftarrow$ Instance parameters
2: $pq$, $best\_assignments \leftarrow$ compute-structures$(d, Y, B, S, p)$
3: $assigned\_clients \leftarrow \emptyset$
4: $full\_centers \leftarrow \emptyset$
5: while $pq$ is not empty and $|assigned\_clients| < B$ do
6:     $j \leftarrow$ dequeue $pq$
7:     if $j \notin assigned\_clients$ then
8:         $\phi_{\max} \leftarrow$ last element from $best\_assignments[j]$
9:         $\phi_{\min} \leftarrow$ first element from $best\_assignments[j]$
10:        RCL$\leftarrow \{j : \phi(j) \leq \phi_{min} - \alpha_a(\phi_{max} - \phi_{min})\}$
11:        Select random $i$ from RCL as long as it is not in $full\_centers$
12:        $X_{i,j} \leftarrow 1$
13:        Add client to $assigned\_clients$
14:        Update $values\_m$, $risk\_v$ with $i, j$
15:        if center $i$ has reached lower bounds in activity measures or in the risk value then
16:            Add $i$ to $full\_centers$
17:            break
18:        end if
19:    end if
20: end while
21: Assign all clients $\notin assigned\_clients$ to the centers with the largest capacity available in the risk threshold $\beta$.
22: return $X$
```

4.3 Local Search Heuristics

Local search heuristics operate on the principle of iteratively exploring the neighborhood of a current solution in the search space to find a new solution that is better according to a given objective function (Taillard, 2023). *The neighborhood* of a solution typically consists of all solutions that have a small, predefined change in their structure from the current solution. The steps taken to move from one solution to another are defined by what is called a *move operator*.

They explore the local space of a given initial solution trying to either improve the objective function value or repair its feasibility of the solution, we two strategies were implemented and tested: (i) A Best Found (BF) strategy that moved to the best improving neighbor and (ii) a First Found (FF) strategy that moved to the first improving neighbor. In our testings, shown in Table 5.4, the FF yield better results, so this is the strategy chosen for the final GRASP implementation.

In this work, some infeasible initial solutions can be a result of the constructive heuristic, because of this, we have developed a Local Search procedure that can handle infeasibility by first repairing the solutions, once solutions are feasible, it moves on improving the solution quality. This procedure is explained below.

4.3.1 Repairing Infeasibilities

As stated previously, the constructive heuristic proposed in this thesis may provide solutions that are not feasible. A solution may be infeasible because of two reasons: a center exceeds the maximum number of allocated BUs, because of the activities' measures surpassing their maximum target, or the risk threshold is exceeded; or the BUs assigned were not enough to complete the minimum required in the activities' measures. To avoid this, the local search applies one of the two following basic moves: **Add**(i) which will add BUs to a center i, that is violating the lower bounds

of the constraints until those bounds are met, while ensuring that the moved BUs do not render their previous centers infeasible, and **Remove**(i) which will remove BUs from a center i that exceeds the upper bounds of the constraints, ensuring that whichever center those BUs end up being reallocated to does not become infeasible as well.

By directing the local search with “AddTo” or “RemoveFrom” sets for centers, we achieve faster computations, as it is a straight-forward to determine to which set does a center is member of, as shown in Algorithm 7. The moves are applied until all centers are feasible, transforming the solution to a feasible one, expressed in Algorithm 8.

Algorithm 7 `compute-add-remove`(Instance, X, Y)

```
Input: Instance of the problem, $X$ decision matrix, $Y$ decision vector
Output: $AddTo$ set of centers to add BUs, $RemoveFrom$ set of centers to remove
    BUs
1: for $i \in Y$ do
2:     if Any value from $X, Y_i$ is greater than the upper bounds from the con-
       straints then
3:         $RemoveFrom \leftarrow i$
4:     else if Any value from $X, Y_i$ is less than the lower bounds from the con-
       straints then
5:         $AddTo \leftarrow i$
6:     end if
7: end for
8: return $AddTo, RemoveFrom$
```

Algorithm 8 `repair-solution`(Instance, X, Y)

```
Input: Instance of the problem, $X$ decision matrix, $Y$ decision vector
Output: $X'$ decision matrix
1: $AddTo, RemoveFrom \leftarrow$ compute-add-remove(Instance, $X$, $Y$)
2: $X'$ copies $X$
3: for $\tilde{i} \in AddTo$ do
4:     while Any value from $X, Y_{\tilde{i}}$ is less than the lower bounds from the constraints do
5:         Allocate the nearest $j$ BU to $\tilde{i}$, removing it from its previous center $i*$
6:         $X'_{i*,j} = 0$
7:         $X'_{\tilde{i},j} = 1$
8:         Update the constraints' states for $\tilde{i}$ and $i*$
9:     end while
10: end for
11: for $\tilde{i} \in RemoveFrom$ do
12:     while Any value from $X, Y_{\tilde{i}}$ is greater than the upper bounds from the constraints do
13:         Move the furthest $j$ BU from $\tilde{i}$, allocating it to its nearest center $i*$
14:         $X'_{\tilde{i},j} = 0$
15:         $X'_{i*,j} = 1$
16:         Update the constraints' states for $\tilde{i}$ and $i*$
17:     end while
18: end for
19: return $X'$
```

4.3.2 Improving objective function value

Once the solution is feasible, the local search can now proceed to improve the objective function value. For this, the following moves are defined:

- **Allocate Other**(i, j): Reassign a BU j from center i to another center $\tilde{i}$, where $\tilde{i} \neq i$, $\iff X_{ij} = 1$.
- **Interchange**$(i, j, \tilde{i}, \tilde{j})$: Swap allocations of BUs j and $\tilde{j}$ such that j is allocated to $\tilde{i}$ and $\tilde{j}$ to i, $\iff X_{ij} = 1 \land X_{\tilde{i}\tilde{j}} = 1$.
- **Deactivate**$(i, \tilde{i})$: Close center i and open an unused center $\tilde{i}$. Assign BUs to $\tilde{i}$ until the lower bounds of the balancing constraints are met, and reallocate the remaining BUs from i to the best-fit centers, $\iff Y_i = 1 \land Y_{\tilde{i}} = 0$.

The Deactivate move reallocates the remaining BUs using the same dictionary with the best assignments to each BU from the allocation phase of the constructive heuristic, as it already has the centers sorted by their distance to the BUs.

4.3.3 Combining Multiple Neighborhoods

As we have designed multiple neighborhoods for the local search phase, it is natural to combine these multiple moves in a cyclical procedure, exploring the local space of a move until it reaches the local optimum value and then it moves on to the next move, until no move improves the solution. This helps us escape local optima and provides a more comprehensive exploration of the solution space. The main steps of the algorithm are are:

1. Start with an Initial Solution: Begin with an initial solution and a set of predefined neighborhood structures. In this case, the initial solution is the solution provided by the constructive heuristic and the set of neighborhoods is the collection of the 3 moves defined previously.

2. Sequential Neighborhood Search: Explore these neighborhoods in a systematic way. For each neighborhood, perform a local search to find a local optimum. In this case, we performed the **Allocate to Other, Interchange** and **Deactivate** moves sequentially in that order, applying the moves iteratively to the result of the

previous move until all three neighborhoods reached their local optimum, doing so cyclically.

3. Neighborhood Change: If an improvement is found in any neighborhood, the process is restarted from the first neighborhood with the improved solution. If no improvement is found in any of the neighborhoods, the process stops, and the best-found solution is returned.

In Algorithm 9, lines 1 to 3 repair the solution using the previously defined algorithms. The cyclical strategy is applied as a while loop from lines 6 to 29, applying the moves in the sequential order show, until no move improves the incumbent solution. A solution is expressed as (X, Y) where X is the decision variable matrix of allocations and Y the decision variable vector of locations.

Algorithm 9 `local-search`(Instance, X, Y)

Input: Instance of the problem, X decision matrix, Y decision vector

Output: X decision matrix, Y decision vector

```
1: if $(X, Y)$ is not feasible then
2:     $X \leftarrow$ repair-solution($(X, Y)$)
3: end if
4: improvement $\leftarrow$ True
5: $z \leftarrow f(X, Y)$
6: while improvement do
7:     improvement = False
8:     $(X', Y') \leftarrow$ Apply Allocate Other to $(X, Y)$ until local optimum is met
9:     $z' \leftarrow f(X', Y')$
10:    if $z' < z$ then
11:        $(X, Y) \leftarrow (X', Y')$
12:        $z = z'$
13:        improvement = True
14:    end if
15:    $(X', Y') \leftarrow$ Apply Interchange to $(X, Y)$ until local optimum is met
16:    $z' \leftarrow f(X', Y')$
17:    if $z' < z$ then
18:        $(X, Y) \leftarrow (X', Y')$
19:        $z = z'$
20:        improvement = True
21:    end if
22:    $(X', Y') \leftarrow$ Apply Deactivate to $(X, Y)$ until local optimum is met
23:    $z' \leftarrow f(X', Y')$
24:    if $z' < z$ then
25:        $(X, Y) \leftarrow (X', Y')$
26:        $z = z'$
27:        improvement = True
28:    end if
29: end while
30: return $(X, Y)$
```

4.4 Parallel computing

As the individual components for the heuristics were chosen due to their performance detailed in Chapter 5, the GRASP metaheuristic was built from these individual algorithms. An advantage of the GRASP metaheuristic is that each individual iteration can be parallelized, as each iteration is unique and independent from the next or previous one. By assigning an iteration to a computer core thread, we can speed up the computation by introducing parallelization with the use of such computer threads, synchronizing the threads only when checking for the best solution found across the multi-threaded computation. This parallelization mechanism was successfully implemented for the GRASP metaheuristic, providing faster results compared to a single-threaded GRASP version, as shown in section 5.5.1.1.

In order to properly implement a multi-threaded version of the GRASP metaheuristic we had to use the computer science concept of locks. A lock, also known as a mutex (derived from mutual exclusion) is a construct that allows us to safely manage different state variables, by not allowing them to be accessed from multiple threads at the same time, preventing race conditions, which occur when two different threads try to modify the same variable at the same time, entering into a "race" of accessing and modifying the variable, ensuring a safe and correct implementation of the algorithm.

Chapter 5

Computational Results

In this chapter we present the results of the computational experience of the proposed algorithm. We highlight the performance advantages of using data structures and an intelligent design of algorithms versus a more naive version of the algorithm. Additionally, we compare the obtained solutions with those provided by Gurobi version 11.0.3, a state-of-the-art solver. We also show the effectiveness of implementing a parallel version of GRASP, reducing running times significantly. Finally, we provide the best parameters to be used in the GRASP metaheuristics as result of a fine-tuning phase.

5.1 Instance generation

To assess our procedure, some problem instances were randomly generated. Throughout the thesis, the Julia programming language is used to carry out all the computations related to the problem. The Julia language was designed to be used in scientific computing contexts, with a simple syntax reminiscent of Python and FORTRAN while providing the performance of C/C++, showing that one can have machine performance without sacrificing readability (Bezanson et al., 2014). From the instance generation, to the modeling of the problem as a mathematical program and the GRASP metaheuristic, every stage of computation was written in Julia.

In order to generate problem instances, several routines are detailed and executed in the following order: first, the coordinates in a Cartesian plane of the BUs and the centers are randomly generated from an interval of 5 to 10,000; second, the d distance matrix is calculated from these coordinates using the Euclidean distance metric, rounding the distances to integers. Third, the T_{ik} parameter is generated, assigning a random k type to each center, we define the L_k and U_k bounds proportional to the the sum of all the centers for each type k. Afterwards, for each activity measure m, a random value is assigned to every individual BU j in the vector v_m^j, from a range of 10 to 30. Once all the BUs have an activity measure value for all the activities, the targets of the measures are calculated as the sum of all the measure values divided by $|S|$ and multiplied by an additional τ parameter to introduce some leeway, with a value of 1. The t_m tolerance parameter is defined as 0.4. Finally, the risk value for every BU j is randomly generated in the 40 to 60 range, with the risk threshold for each center i defined as the sum of all the individual values divided by $|S|$ and multiplied by the same τ parameter $((\sum_{j \in B} \frac{r_j}{S})\tau)$.

All of the instances are written using an HDF5 standard compliant file format, in order to allow serialization and data sharing between programming languages and computing platforms, as long as the programming language has a valid HDF5 parser library to read the instances and the solutions (Koranne, 2011).

Three sizes of instances were generated, the BUs and centers' coordinates were generated from a range of [5, 10,000] in the Cartesian plane, with the following configuration:

Table 5.1: Sizes of the generated instances

Size	$\|B\|$	$\|S\|$	p	Number of instances
1	625	62	32	20
2	1250	155	62	20
3	2500	325	125	10

5.2 Exact solutions

Algebraic modeling languages (AMLs) are domain specific computer languages that are used to describe and solve mathematical optimization models with a similar syntax to the mathematical notation of optimization problems. This makes it possible to define optimization problems in a very clear and understandable manner. This is facilitated by specific language components such as sets, indices, algebraic expressions, sparse indices, variables for handling data, and constraints with arbitrary names (Kallrath, 2004).

Instead of directly solving those problems, an AML calls relevant external algorithms to optimize the problem. These programs, known as solvers, use exact algorithms designed to handle a wide range of optimization problems, such as linear programs, integer programs, and quadratic programs, for example. The algorithms use mainly branch and bound, and algorithmic strategies depending on the structure of the structure of the problem and the solver itself. These algorithms can also be used to implement more advanced methods such as branch and cut, branch and price, or cutting planes.

By using an AML to represent the mathematical model described in Chapter 3, we can feed the resulting abstraction of the model to an optimization software package.

In this specific case, the algebraic modeling language used is the JuMP.jl package from the Julia language. JuMP enables developers to use a very concise syntax that enables a near one-to-one mapping of the mathematical notation to the language features (Dunning et al., 2017), and the mathematical solver used is the Gurobi Optimizer, a commercial solver that uses state-of-the-art optimization algorithms and techniques to solve problems, providing better results than open source solvers can (Gurobi Optimization, LLC, 2023). We briefly tested the CPLEX and HiGHS optimizers but decided to use Gurobi based on the preliminary results when comparing

to the mentioned solvers.

Using the Julia language version 1.10.4 and Gurobi version 11.0.3, we modeled the problem using JuMP version 1.22.2 and introduced the corresponding model and instance to the solver, setting two time limit for the optimization process of 1800 seconds and 7200 seconds, varying on the size of the instance. The results reported are the total optimization time of the algorithm, the best integer solution found, the best lower bound, as well as the relative optimality gap or MIP (Mixed Integer Programming) gap. According to Gurobi Optimization, LLC (2023), "the MIP gap is defined as $|z_P - z_D|/|z_P$, where z_P is the primal objective bound (i.e., the incumbent objective value, which is the upper bound for minimization problems), and z_D is the dual objective bound (the lower bound for minimization problems)". The MIP gap for the termination of Gurobi's algorithm is left as the default value of 0.01%, a hundredth part of 1%.

For Size 1, the results are shown in Table 5.2. As appreciated, out of the 20 instances, no solutions were proven to be optimal within the time limit of 1800 seconds (half an hour). The average relative optimality gap (MIP gap) percentage of the solutions is 0.28%, proving that even for the smallest instances, the $\mathcal{NP}$ nature of the problem is impacting the performance of exact solvers, although the gap to optimality is minimal. This is useful when comparing to the results of the heuristics, as we can compare to near optimal results provided by the best bound.

As shown in the Appendix in tables A.1 and A.3, for Sizes 2 and 3, no optimal solution was found within half and hour and 2 hours, the respective time limits for the instances. The average MIP gap percentage of Size 2 is 0.45% and for Size 3 is 0.18%. These values represent almost optimality, in most cases, the optimal solution was reached and the algorithm spent the rest of the time proving optimality. Since we are dealing with an $\mathcal{NP}$-hard problem, Gurobi struggles to prove that the best solution is the optimal one, spending computational time, showcasing the usefulness of the heuristics.

Table 5.2: Exact solver solutions for Size 1

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	1800	464274	462154.44	0.46
2	1800	475106	474267.15	0.18
3	1800	451907	451740.15	0.04
4	1800	460877	460520.42	0.08
5	1800	469190	468298.94	0.19
6	1800	491611	490305.18	0.27
7	1800	454550	453366.41	0.26
8	1800	462113	460970.18	0.25
9	1800	488971	487836.91	0.23
10	1800	471810	467610.27	0.89
11	1800	468195	466413.29	0.38
12	1800	461823	461040.14	0.17
13	1800	468480	468091.44	0.08
14	1800	459301	459097.84	0.04
15	1800	483950	479672.39	0.88
16	1800	462365	461967.27	0.09
17	1800	462745	462126.53	0.13
18	1800	455692	454556.37	0.25
19	1800	461881	458774.94	0.67
20	1800	465070	464547.88	0.11

5.3 Constructive heuristics

In this section the results of the different strategies that compose the constructive heuristic are presented. As a reminder, the heuristic has two main phases: location and allocation. For location, we defined that we can use either a p-dispersion based heuristic or the Partial Linear Programming (PLP henceforth) relaxation approach. For allocation, we can use the opportunity cost algorithm that calculates the whole opportunity cost matrix or the algorithm that incorporates data structures such as the opportunity cost priority queue, referring to each as “matrix” or “queue” algorithms, respectively. All instances of Sizes 1 and 2 were solved with all the possible combinations of location and allocation strategies. In Table 5.3 we can see the averages of the run-time of the algorithms and the number of constraints violated by the resulting solutions, which gives us an insight on how far away are the solutions from feasibility. ε denotes a time equal or shorter than 0.01 seconds. The Total Time column is the sum of the location and allocation phases.

Table 5.3: Comparison among the Constructive Heuristic strategies (averages)

Inst. Size	Location Strat.	Location Time (s)	Allocation Time (s)		Total Time (s)		# of violated constraints before repair	
			Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.	Matrix Alloc.	Queue Alloc.
1	P-disp	0.02	0.22	ε	0.24	0.04	43.75	5.4
	Relaxation	1.00	0.31	ε	1.31	1.01	49.75	4.6
2	P-disp	0.03	1.60	0.02	1.63	0.05	81.55	0.15
	Relaxation	11.61	1.63	0.02	13.23	11.79	92.01	0.30

The p-dispersion location strategy outperforms the partial linear programming relaxation strategy in execution time, therefore we will use it as the building block for our GRASP metaheuristic. Meanwhile, the queue allocation strategy consistently outperforms the matrix allocation strategy in both run time and feasibility of the created solution. As seen in the table, as the instances get larger, the queue allocation algorithm almost always provides feasible solutions, reducing the need of using the

repair algorithm in the local search procedure.

5.4 Local search heuristics

As defined in the previous chapter, there are three moves to perform in the local search procedure: **Allocate to Other, Interchange** and **Deactivate**. These three moves were used together in a cyclic local search, in the same order, applying the moves iteratively to the result of the previous move until all three neighborhoods reached their local optimum. Local search heuristics can guide their search by either moving to the best solution found across the neighborhood or move to the first solution that improves the objective function value, without exploring all the alternatives. We define these two strategies as Best Found (BF) or First Found (FF). Applying the local search with either BF or FF strategy to the solutions produced by the constructive heuristic we present the following results:

Table 5.4: Evaluation of local search strategies (averages)

Inst. Size	Strategy	% Rel. Optimality Gap	Time (s)
1	BF	10.88	0.89
	FF	15.49	0.52
2	BF	10.73	10.98
	FF	10.52	6.37

As appreciated, the Best Found strategy provides slightly better results on average in the optimality gap to the best bound in the Size 1 instances, but requires a longer time. For Size 2, the gaps are comparable, and times are shorter for First Found. Therefore, we chose the First Found strategy to be used in the GRASP search procedure, preferring shorter run-times.

5.5 GRASP

With the p-dispersion and opportunity cost queue enhanced strategies chosen for our constructive heuristic and the First Found strategy along with the cyclical mechanism for the search heuristic, we can now integrate and adapt these building blocks into the Greedy Randomized Adaptive Search Procedure metaheuristic, defining their respective modifications in Chapter 4.

GRASP requires the α_l and α_a quality threshold parameters for the RCL as well as the number of iterations that it will run for. Moreover, as our GRASP can be parallelized, the number of computer core threads used to parallelize the computation needs to be calibrated as well for maximum performance.

5.5.1 Fine-tuning of GRASP parameters

5.5.1.1 Number of threads

The first parameter that needs to be calibrated is the number of threads that the GRASP metaheuristic will run on. We defined a basic configuration with $\alpha_l = 0.1, \alpha_a = 0.1$ and 50 iterations. As the number of threads depends on the number of cores of each computer's CPU, we decided to run from 1 to 16 threads, as that is the maximum number of threads in the computing platform in which we ran the experiments. We obtained the results shown in Figures 5.1 and 5.2.

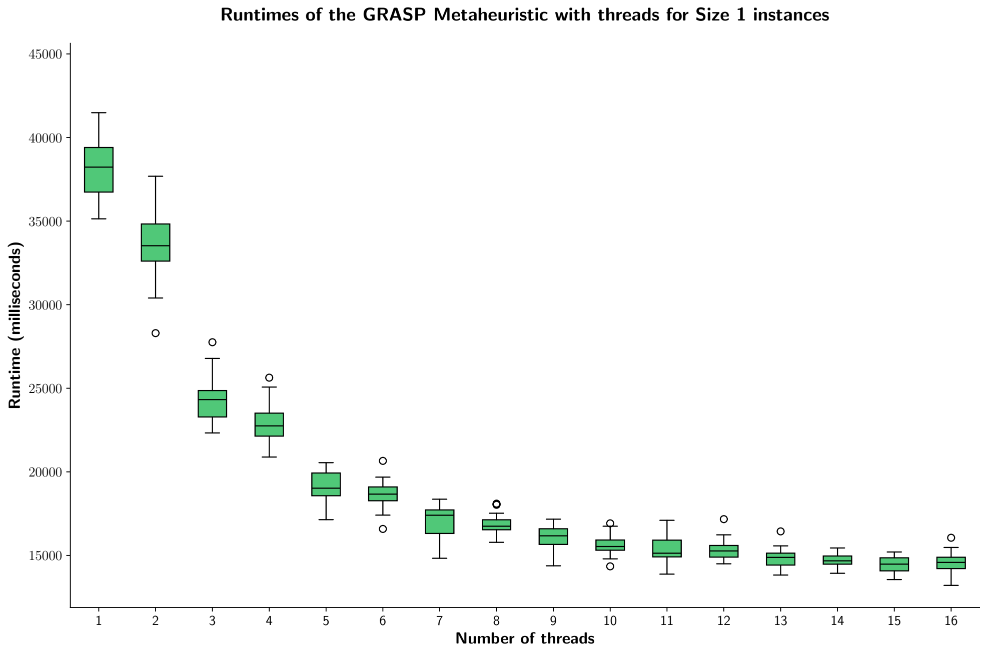


FIGURE 5.1: Boxplot of the GRASP's run-time for Size 1 instances

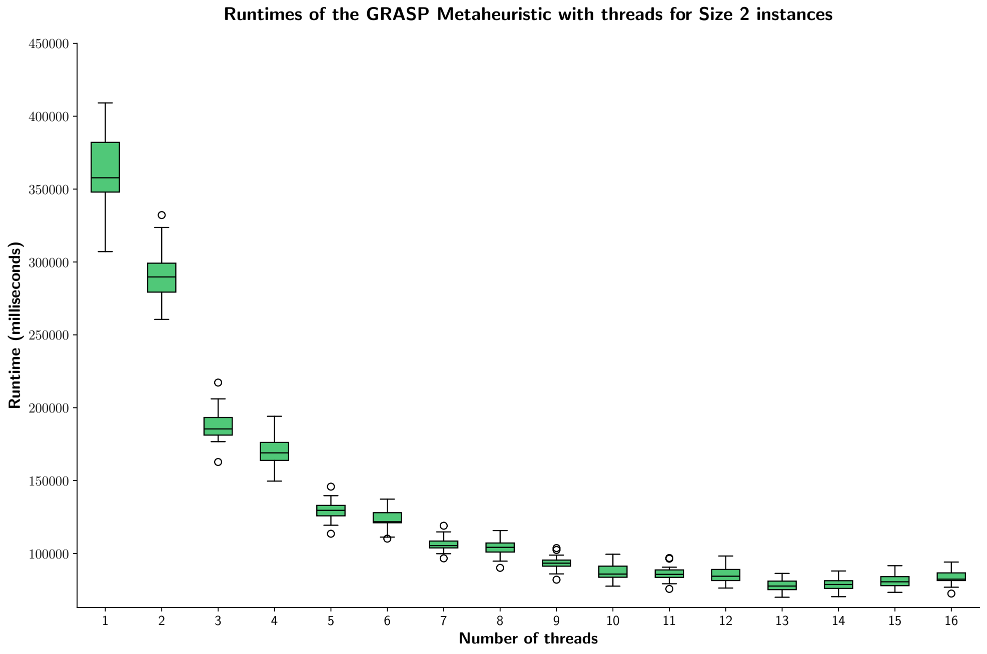


FIGURE 5.2: Boxplot of the GRASP's run-time for Size 2 instances

Statistical hypothesis testing was performed on the results to determine the optimal amount of threads to use as well as to check if that number of threads had any impact on the objective function value, which is something undesired.

The full extent of the statistical tests is present in the Appendix, more specifically section A.5, the main conclusions extracted were that: increasing the number of threads generally improves performance, with the most substantial gains observed when moving from 1 to 14 threads; the number of threads for minimizing run-time appears to be around 14 or 15; performance plateaus or shows minor reductions after 14 threads, increasing more the number of threads does not have a significant impact on the objective value, as evidenced by the ANOVA (Analysis of Variance) on objective values. After 14 threads, the overhead of communication between the computing processes, the scheduler of the operating system and the memory requirements negate any further improvement in the computation time of the heuristic.

A very interesting thing to note as well is that the size of the problem impacts the performance per thread. For Size 1, we minimize run-times using 16 threads, but due to a heavier computing load for Size 2, we find a better run-time with 14 rather than 16 threads. This can be explained using Amdahl's law (Amdahl, 1967), a computer architecture law that is used in parallel computing to explain and predict the speedup of a process if ran in multiple processors or cores. A heavier load may also impact the task scheduler of the host operating system, larger instances may also incur more memory costs, having a larger overhead, and in this case, making the increasing thread count a penalty from 15 threads onward.

When testing for Size 3 instances, a high variability when running different numbers of threads was found. In some runs, 8 threads provided the best times, whereas for the same instance, if ran with 12 threads at a later point in time, it ran faster than 8 threads, but when trying to replicate even later, the 12 threads were worse than the 8 threads. This was a hindrance on testing and calibrating the number of threads for larger instances, but was an interesting finding, as it confirms

that the size of the instance does impact the best number of threads, pointing to the computing overhead of more complex instances. The reasons of this inconsistent behavior were not made clear, so future work could tackle this issue.

It is important to note that Gurobi will use all available CPU cores by default, making it parallel as well.

5.5.1.2 Calibration of α parameters

Next we have to calibrate the α parameters as well as the number of iterations. We tested the following combinations for all instances of Sizes 1 and 2: $\alpha_l \in \{0.1, 0.3, 0.5, 0.7\}$, $\alpha_a \in \{0.1, 0.3, 0.5, 0.7\}$ and $iters \in \{25, 50, 75, 100\}$, giving us a total of 64 combinations to run. In Tables A.28 and A.29 we present the statistical description of running the GRASP for all the combinations, solving 10 instances of Sizes 1 and 2 each.

We carried out one-way Analysis of Variance (ANOVA) on the three parameters and whether they have an impact or not in the run-time of the metaheuristic as well as the objective function value. For the iterations parameter, the ANOVA test on both results revealed statistically significant differences across the different levels of iterations, meanwhile both α did show statistically significant differences across their different levels for instances of Size 1, but the same behavior was not seen in instances of Size 2. We also carried out a Factorial ANOVA to statistically control for variables and the interactions they may have between each other, obtaining varied results for both instance sizes. As the number of iterations increases, so does the run-time and the objective function improves slightly, but the results for α are inconclusive, as different results are observed across the different sizes, they may also be due to the factorial nature of this test. Finally, we performed Tukey's Honest Significant Differences testing, showing that α_l and α_a have impact on the objective function value for instances of Size 1, but not for instances of Size 2. Therefore, we decided

to settle for $\alpha_l = 0.1, \alpha_a = 0.3$ and $iters = 100$. Both α have good enough averages across all of the combinations for the two instance sizes and the number of iterations decreases our relative optimality gap, incurring a time penalty that is acceptable when comparing to the run-time of Gurobi.

5.5.2 GRASP performance

With $\alpha_l = 0.1, \alpha_a = 0.3$, $iters = 100$ and the number of threads to be used set to 14, we can now properly run the GRASP metaheuristic and obtain results for all of the instances. In this final experiment, we also used 10 instances of Size 3 to evaluate both the metaheuristic as well as Gurobi to their limit. Due to thermal throttling and inconsistent core usage, we had to reduce the number of iterations to 25 in the instances of Size 3. In this case, Gurobi was set a time limit of two hours, and the results can be found in the Appendix, on Table A.3

The most important table that can condense the entire work of this thesis is the following:

Table 5.5: Results of the Fine-tuned GRASP with 14 threads

Instance Size	% Optimality Gap			Speedup Factor vs Gurobi		
	Min	Avg	Max	Min	Avg	Max
1	5.22	6.71	8.56	82.68	124.74	144
2	6.33	7.55	9.91	11.05	12.25	13.45
3	7.38	8.34	8.97	3.84	4.26	4.95

The optimality gap to Gurobi's best bound z_D is calculated with the same formula defined by Gurobi Optimization, LLC (2023), which in this case z_P is the value provided by the GRASP metaheuristic.

As the table shows, the GRASP metaheuristic runs several orders of magnitude

faster than Gurobi while providing solutions that are a single digit of relative difference with respect to the solutions found by Gurobi. For all the tables of results, when comparing to Gurobi, the relative difference is how worse are the solutions found by the heuristics.

For Instance Size 1, GRASP runs more than 100 times faster than Gurobi, providing solutions that have a 6.71% of optimality gap.

For Instance Size 2, the speedup is not as dramatic but it is still on average 12 times faster, with a slightly worse average optimality gap of 7.55%.

Finally, for Instance Size 3, the speedups are still there, reduced, but still 4 to 5 times faster than Gurobi, with the optimality gap still a digit away. In this final instance size, Gurobi had several out-of-memory errors when working with the instances, so multiple retries had to be done to solve the problem until it reached the specified timeout.

The full results are present in the following tables:

Table 5.6: GRASP results on Size 1 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	1800	464274	0.46	462154.4	15.32	501167	7.78	117.49
2	1800	475106	0.18	474267.2	13.33	507463	6.54	135.03
3	1483	451907	0.04	451740.2	13.48	491261	8.04	133.53
4	1800	460877	0.08	460520.4	13.63	498939	7.70	132.06
5	1800	469190	0.19	468298.9	14.24	496673	5.71	126.40
6	1800	491611	0.27	490305.2	14.90	522147	6.10	120.81
7	1800	454550	0.26	453366.4	12.50	495818	8.56	144.00
8	1800	462113	0.25	460970.2	12.54	497735	7.39	143.54
9	1800	488971	0.23	487836.9	13.23	517519	5.74	136.05
10	1800	471810	0.89	467610.3	21.77	506917	7.75	82.68
11	1800	468195	0.38	466413.3	14.86	507179	8.04	121.13
12	1800	461823	0.17	461040.1	14.91	489697	5.85	120.72
13	1800	468480	0.08	468091.4	13.93	499595	6.31	129.22
14	976	459301	0.04	459097.8	14.11	499018	8.00	127.57
15	1800	483950	0.88	479672.4	14.35	506120	5.23	125.44
16	1384	462365	0.09	461967.3	13.12	496302	6.92	137.20
17	1800	462745	0.13	462126.5	14.88	492972	6.26	120.97
18	1800	455692	0.25	454556.4	14.11	480927	5.48	127.57
19	1800	461881	0.67	458774.9	14.51	485495	5.50	124.05
20	1800	465070	0.11	464547.9	14.88	490130	5.22	120.97

Table 5.7: GRASP results on Size 2 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	1800	949005	0.44	944849.5	145.24	1022786	7.62	12.39
2	1800	975979	0.92	967012.3	140.51	1043541	7.33	12.81
3	1800	988670	0.67	982048.4	159.46	1082240	9.26	11.29
4	1800	974186	0.29	971396.9	147.49	1049803	7.47	12.20
5	1800	958455	0.05	957950.0	135.87	1039603	7.85	13.25
6	1800	964088	0.22	961976.2	139.55	1036826	7.22	12.90
7	1800	968567	0.56	963149.9	143.17	1039149	7.31	12.57
8	1800	986315	0.32	983207.5	133.83	1049604	6.33	13.45
9	1800	951731	0.45	947419.2	139.09	1025924	7.65	12.94
10	1800	999440	0.84	991050.3	159.94	1100104	9.91	11.25
11	1800	961532	0.70	954783.3	137.66	1022340	6.61	13.08
12	1800	942864	0.20	940956.3	152.63	1008988	6.74	11.79
13	1800	1012710	0.32	1009464.0	162.79	1114006	9.38	11.06
14	1800	987762	0.28	985021.3	153.46	1058794	6.97	11.73
15	1800	961675	0.48	957101.4	148.60	1030436	7.12	12.11
16	1800	990828	0.70	983915.7	160.70	1057756	6.98	11.20
17	1800	992812	0.13	991539.1	145.73	1067347	7.10	12.35
18	1800	980761	0.64	974474.1	141.49	1056380	7.75	12.72
19	1800	973562	0.28	970867.4	136.73	1040261	6.67	13.16
20	1800	972605	0.45	968213.7	154.64	1049519	7.75	11.64

Table 5.8: GRASP results on Size 3 instances

	Gurobi				GRASP			
Instance	Time	Objective Function	%Optimality gap	Best Bound	Time	Objective Function	%Optimality gap	Speedup Factor
1	7200	883693	0.29	881117.8	1746.70	959509	8.17	4.12
2	7200	893408	0.15	892105.5	1655.75	963215	7.38	4.35
3	7200	882602	0.04	882275.8	1452.81	966988	8.76	4.96
4	7200	913347	0.26	910961.7	1748.65	1000684	8.97	4.12
5	7200	895581	0.21	893657.9	1633.76	975003	8.34	4.41
6	7200	896157	0.18	894518.5	1873.31	966452	7.44	3.84
7	7200	893684	0.13	892487.9	1626.62	979920	8.92	4.43
8	7200	895125	0.28	892662.6	1641.34	974882	8.43	4.39
9	7200	877510	0.12	876485.6	1836.52	957017	8.41	3.92
10	7200	892898	0.14	891652.1	1657.39	975116	8.56	4.34

Chapter 6

Conclusions

6.1 Main findings and conclusions

By using intelligent algorithms as well as efficient data structures, we solved the Territorial Design Problem for microfinancial institutions using a GRASP metaheuristic that proved to be competitive with commercial exact optimization software, being several orders of magnitude faster in the execution time while providing solutions that are just slightly worse than the ones found by Gurobi, which are near-optimal.

We developed a mathematical model based on previous work by López, Ekin, Mediavilla, and Jimenez (2020), generalizing it and making it easier to solve in general, building an instances generator program for this model. We successfully implemented this mathematical model in a modeling language in order to be solved with an optimization solver, in this case Gurobi, which is a commercial software that we could use through the use of an academic license.

Using the p-dispersion heuristic for location and opportunity cost with the usage of efficient data structures for allocation we can offer the best performance in feasibility and speed, as well as using a local search heuristic to repair to feasibility the solutions constructed and improve these solutions. These two heuristics were adapted and integrated within the GRASP using a value-based restricted candidate

list to promote diversification. We also reduced the time complexity of the first allocation heuristic by introducing several data structures: a priority queue, a dictionary, arrays and matrices to remove repeated computations as well as partially evaluating the solution built, instead of computing everything every time.

By also using parallel computing, we reduced the times of the metaheuristic, exploiting its parallelizable nature and using modern computing infrastructure that maximizes the performance of today's processors, although for larger instances we found erratic loads in the CPU and the threads, which could be due to thermal throttling, garbage collection of the programming language or frequent cache misses which led to memory access slowdowns.

For small size instances, the heuristics sometimes struggle to find feasible solutions, for larger size instances, feasibility is achieved for all instances. However, once the heuristics are integrated in the GRASP framework, feasibility is always reached.

The results found during the computational experiments enabled us to take informed decisions while building the metaheuristic framework, allowing us to fine-tune the parameters of the GRASP algorithm to provide results with a good balance of runtime and objective function value. GRASP's solutions closely rival Gurobi's but are significantly faster.

6.2 Future work

Instead of using risk as a given parameter of the problem, we could model it as an uncertainty measure of a probability distribution that describes the probability of clients defaulting on their loans and financial services provided. By doing so, we can introduce a stochastic component to the problem, turning it into a stochastic optimization problem, in which we must generate scenarios and use estimations and samples to solve it using a robust framework, according to Spall (2003).

As stated back in Chapter 3, the problem studied in this thesis does not consider contiguity constraints. By incorporating contiguity constraints we can model the problem as a graph, more so than the current problem, and use graph theory algorithms to ensure connected territories as graphs.

We can also try to profile and run the metaheuristic controlling for certain variables to determine the cause of the extreme spikes and valleys in CPU load and temperature when running the multi-threaded GRASP in instances of Size 3. By profiling the code, its memory allocations and setting specific parameters such as voltage of the CPU and keeping track of CPU load and temperature, we can pinpoint where is the issue, allowing us to have a clear answer and a solution to the inconsistent speeds of the GRASP implementation.

Both completely change the nature of the problem as well as requiring that the solution methods are adapted to account for these changes. Thus, they provide interesting avenues of research and future work.

The local search heuristic was improved by using a cyclical approach, laying the groundwork for using a Variable Neighborhood Search metaheuristic (Hansen et al., 2010), which has been successfully implemented for territorial design problems (Quevedo-Orozco and Ríos-Mercado, 2015).

Finally, we can explore the use of other strategies to solve the problem, such as using different metaheuristics: the mentioned VNS, Tabu Search, or Scatter Search. We could even hybridize the heuristic with mathematical programming, by partially solving the problem with the heuristics and then passing it to the solver as a warm start for its algorithms.

Appendix A

Appendix

A.1 Exact solutions tables of results

For the results of Size 1, the table is found in Chapter 5.

Table A.1: Exact solver solutions for Size 2 (Instances 1-10)

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	1800	949005	944849.51	0.44
2	1800	975979	967012.29	0.92
3	1800	988670	982048.42	0.67
4	1800	974186	971396.86	0.29
5	1800	958455	957950.01	0.05
6	1800	964088	961976.21	0.22
7	1800	968567	963149.90	0.56
8	1800	986315	983207.46	0.32
9	951731	951731	947419.24	0.45
10	1800	999440	991050.29	0.84

Table A.2: Exact solver solutions for Size 2 (Instances 11-20)

Instance	Time (seconds)	z	Best Bound	% MIP Gap
11	1800	961532	954783.34	0.70
12	1800	942864	940956.26	0.20
13	1800	1012710	1009464.27	0.32
14	1800	987762	985021.26	0.28
15	1800	961675	957101.37	0.48
16	1800	990828	983915.69	0.70
17	1800	992812	991539.11	0.13
18	1800	980761	974474.12	0.64
19	1800	973562	970867.41	0.28
20	1800	972605	968213.68	0.45

Table A.3: Exact solver solutions for Size 3

Instance	Time (seconds)	z	Best Bound	% MIP Gap
1	7200	883693	881117.84	0.29
2	7200	893408	892105.54	0.15
3	7200	882602	882275.84	0.04
4	7200	913347	910961.69	0.26
5	7200	895581	893657.94	0.21
6	7200	896157	894518.51	0.18
7	7200	893684	892487.89	0.13
8	7200	895125	892662.59	0.28
9	7200	877510	876485.60	0.12
10	7200	892898	891652.12	0.14

A.2 Constructive heuristic tables of results

A very interesting thing to notice is that due to the just-in-time compilation of the Julia language, the first time a function is called, it is also compiled. For all sizes, the 10th instance was the first to be solved, thus all the times for Instance 10 include the compilation time, therefore all times will show up as outliers when compared to the other solutions' times. Still, the compilation time is minimal and has no major impact on results.

The ε parameter denotes a time shorter than 0.1 seconds. The location phase time is reported as 1.0 seconds because that is the minimum time reported by the optimizer in the Partial Linear Programming Relaxation approach.

A.2.1 Instances of Size 1

Table A.4: Results of constructive heuristics under Opp. Queue and PLP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	1	ε	False	4
2	1	ε	False	3
3	1	ε	False	3
4	1	ε	False	3
5	1	ε	False	3
6	1	ε	False	3
7	1	ε	False	3
8	1	ε	False	7
9	1	ε	False	4
10	1	0.27	False	3
11	1	ε	False	7
12	1	ε	False	6
13	1	ε	False	5
14	1	ε	False	6
15	1	ε	False	3
16	1	ε	False	10
17	1	ε	False	3
18	1	ε	False	6
19	1	ε	False	6
20	1	ε	False	4

Table A.5: Results of constructive heuristics under Opp. Queue and p-dP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	ε	False	6
2	ε	ε	False	8
3	ε	ε	False	7
4	ε	ε	False	7
5	ε	ε	False	5
6	ε	ε	False	3
7	ε	ε	False	3
8	ε	ε	False	6
9	ε	ε	False	3
10	0.55	0.27	False	6
11	ε	ε	False	3
12	ε	ε	False	6
13	ε	ε	False	9
14	ε	ε	False	6
15	ε	ε	False	6
16	ε	ε	False	3
17	ε	ε	False	3
18	ε	ε	False	6
19	ε	ε	False	5
20	ε	ε	False	7

Table A.6: Results of constructive heuristics under Opp. Matrix and PLP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	1	0.37	False	45
2	1	0.36	False	51
3	1	0.17	False	46
4	1	0.36	False	49
5	1	0.17	False	50
6	1	0.40	False	49
7	1	0.17	False	46
8	1	0.46	False	49
9	1	0.16	False	43
10	1	0.71	False	51
11	1	0.50	False	53
12	1	0.21	False	55
13	1	0.43	False	50
14	1	0.22	False	54
15	1	0.21	False	48
16	1	0.38	False	53
17	1	0.16	False	54
18	1	0.38	False	53
19	1	0.19	False	49
20	1	0.18	False	47

Table A.7: Results of constructive heuristics under Opp. Matrix and p-dP for Size 1 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	0.21	False	42
2	ε	0.12	False	41
3	ε	0.12	False	38
4	ε	0.12	False	47
5	ε	0.15	False	37
6	ε	0.13	False	49
7	ε	0.15	False	44
8	ε	0.15	False	41
9	ε	0.17	False	43
10	0.55	0.68	False	44
11	ε	0.25	False	40
12	ε	0.20	False	43
13	ε	0.24	False	46
14	ε	0.23	False	49
15	ε	0.24	False	45
16	ε	0.20	False	47
17	ε	0.20	False	44
18	ε	0.17	False	43
19	ε	0.23	False	42
20	ε	0.38	False	50

A.2.2 Instances of Size 2

Table A.8: Results of constructive heuristics under Opp. Queue and PLP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	12	ε	True	0
2	11	ε	True	0
3	15	ε	True	0
4	13	ε	True	0
5	10	ε	False	3
6	10	ε	True	0
7	11	ε	True	0
8	13	ε	True	0
9	11	ε	True	0
10	15	0.29	True	0
11	11	ε	True	0
12	12	ε	True	0
13	12	ε	True	0
14	11	ε	True	0
15	12	ε	True	0
16	11	0.05	False	3
17	11	ε	True	0
18	12	ε	True	0
19	11	ε	True	0
20	11	ε	True	0

Table A.9: Results of constructive heuristics under Opp. Queue and p-dP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	ε	True	0
2	ε	0.01	False	1
3	ε	ε	True	0
4	ε	ε	True	0
5	ε	ε	True	0
6	ε	ε	True	0
7	ε	ε	False	2
8	ε	ε	True	0
9	ε	ε	True	0
10	0.56	0.28	True	0
11	ε	ε	True	0
12	ε	ε	True	0
13	ε	ε	True	0
14	ε	ε	True	0
15	ε	ε	True	0
16	ε	ε	True	0
17	ε	0.01	True	0
18	ε	ε	True	0
19	ε	ε	True	0
20	ε	ε	True	0

Table A.10: Results of constructive heuristics under Opp. Matrix and PLP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	12	1.57	False	88
2	11	1.53	False	103
3	15	1.5	False	98
4	12	1.92	False	93
5	10	1.38	False	81
6	10	2.02	False	94
7	11	1.1	False	96
8	13	1.37	False	91
9	11	1.75	False	99
10	15	1.92	False	98
11	11	1.42	False	92
12	12	2.13	False	85
13	12	1.16	False	101
14	10	1.69	False	87
15	12	1.64	False	82
16	11	2.0	False	101
17	10	1.64	False	83
18	12	1.28	False	85
19	11	2.08	False	89
20	11	1.52	False	94

Table A.11: Results of constructive heuristics under Opp. Matrix and p-dP for Size 2 instances

Instance	Time Location (s)	Time Allocation (s)	Feasibility	Constraints Violated
1	ε	1.5	False	87
2	ε	1.79	False	78
3	ε	1.47	False	80
4	ε	1.77	False	79
5	ε	1.25	False	84
6	ε	1.56	False	80
7	ε	1.87	False	69
8	ε	1.64	False	76
9	ε	1.07	False	80
10	0.54	1.84	False	90
11	ε	1.6	False	73
12	ε	1.6	False	89
13	ε	1.73	False	86
14	ε	1.6	False	73
15	ε	1.6	False	81
16	ε	1.56	False	84
17	ε	1.72	False	90
18	ε	1.36	False	80
19	ε	1.79	False	81
20	ε	1.71	False	91

A.3 Local search heuristic tables of results

The ε parameter denotes a time shorter than 0.1 seconds, LS Cycles is the number of times the Local Search was performed in a cyclic way, first performing a move, moving to another one when the local optimum was reached, while any move presented an improvement over the incumbent best solution. Instances with a relative optimality gap of 100% are instances that could not be repaired to feasibility by the heuristic.

A.3.1 Instances of Size 1

Table A.12: Local Search results with Matrix, PLP and BF Strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.43	0.7	4	83.28	10.54
2	False	1.7	ε	0	0	100
3	True	0.67	0.84	4	132.84	11.75
4	False	1.87	ε	0	0	100
5	True	0.6	0.65	4	63.16	16.73
6	False	2.18	ε	0	0	100
7	False	1.71	ε	0	0	100
8	False	1.81	ε	0	0	100
9	True	0.65	0.72	5	55.01	15.14
10	True	0.63	0.75	5	59.69	7.55
11	False	1.81	ε	0	0	100
12	False	2.18	ε	0	0	100
13	True	0.65	0.56	5	51.65	10.14
14	True	0.67	0.56	4	42.0	15.37
15	True	0.88	0.69	4	90.31	7.66
16	True	0.62	0.67	5	64.56	6.36
17	True	1.02	0.72	5	80.69	12.0
18	True	1.22	1.09	7	151.49	10.09
19	True	1.08	0.72	5	77.63	8.1
20	False	1.46	ε	0	0	100

Table A.13: Local Search results with Queue, PLP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.42	0.94	6	97.14	10.35
2	True	0.14	1.14	6	155.57	9.84
3	True	0.13	1.27	5	137.57	12.14
4	True	0.1	0.86	4	88.66	15.51
5	True	0.05	1.07	4	86.66	11.14
6	True	0.1	0.81	5	79.8	8.76
7	True	0.14	1.49	4	104.18	10.9
8	False	0.37	ε	0	0	100
9	True	0.35	1.09	5	92.59	14.14
10	False	0.5	ε	0	0	100
11	False	0.83	ε	0	0	100
12	False	0.52	ε	0	0	100
13	True	0.36	0.68	5	71.65	9.05
14	True	0.17	0.77	5	53.07	16.4
15	True	0.24	1.1	7	108.33	10.2
16	False	0.79	ε	0	0	100
17	True	0.09	0.94	5	86.54	12.54
18	False	0.8	ε	0	0	100
19	True	0.08	0.74	4	65.49	12.54
20	True	0.15	0.77	4	91.43	12.3

Table A.14: Local Search results with Matrix, p-dP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.75	1.08	4	94.72	11.93
2	False	2.1	ε	0	0	100
3	True	0.32	0.57	3	47.01	11.82
4	True	0.22	0.58	5	28.64	9.93
5	True	2.05	0.6	5	36.76	10.21
6	True	0.4	1.0	5	81.2	6.34
7	True	0.32	0.64	4	34.45	11.25
8	False	1.57	ε	0	0	100
9	True	0.35	0.77	5	35.12	8.09
10	False	2.13	ε	0	0	100
11	False	1.48	ε	0	0	100
12	True	0.46	0.71	5	38.0	11.95
13	True	0.65	1.01	7	56.05	8.61
14	True	0.53	1.12	5	57.47	12.08
15	True	0.48	1.06	7	74.81	7.98
16	True	0.41	0.72	4	58.18	9.78
17	True	0.58	1.0	4	76.76	11.05
18	True	0.45	0.81	4	51.16	9.02
19	True	0.53	0.76	4	57.19	16.88
20	True	2.58	0.77	5	69.58	5.34

Table A.15: Local Search results with Queue, p-dP and BF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	False	0.43	ε	0	0	100
2	False	0.47	ε	0	0	100
3	True	0.22	0.96	4	100.07	10.06
4	True	0.17	0.76	4	65.4	11.95
5	True	0.23	0.99	4	87.99	6.89
6	True	0.14	1.29	5	108.61	7.48
7	True	0.07	0.88	4	75.35	8.7
8	True	0.23	0.8	4	65.71	8.59
9	True	0.13	1.22	4	94.89	8.65
10	False	0.71	ε	0	0	100
11	False	0.45	ε	0	0	100
12	True	0.18	1.02	6	69.52	10.82
13	False	0.54	ε	0	0	100
14	True	0.2	1.01	3	82.39	14.51
15	True	0.24	1.28	5	95.25	11.87
16	False	0.43	ε	0	0	100
17	True	0.12	1.09	5	101.59	13.55
18	True	0.23	1.11	4	83.32	9.87
19	True	0.07	1.1	4	105.36	16.18
20	False	0.5	ε	0	0	100

Table A.16: Local Search results with Matrix, PLP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.47	0.58	6	86.21	9.11
2	False	1.72	ε	0	0	100
3	True	0.65	0.65	6	136.82	10.24
4	False	1.91	ε	0	0	100
5	True	0.56	0.57	5	71.45	12.49
6	False	2.22	ε	0	0	100
7	False	1.68	ε	0	0	100
8	False	1.86	ε	0	0	100
9	True	0.62	0.4	4	51.49	17.07
10	True	0.64	0.53	5	58.32	8.34
11	False	1.82	ε	0	0	100
12	False	2.18	ε	0	0	100
13	True	0.63	0.4	5	50.6	10.76
14	True	0.65	0.39	4	47.14	12.31
15	True	0.88	0.61	6	83.38	11.02
16	True	0.68	0.42	4	64.29	6.52
17	True	0.97	0.4	4	80.89	11.9
18	True	1.28	0.59	5	154.55	8.99
19	True	1.04	0.51	5	72.94	10.53
20	False	1.45	ε	0	0	100

Table A.17: Local Search results with Queue, PLP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.2	0.41	4	69.72	22.82
2	True	0.11	0.28	3	75.38	38.13
3	True	0.09	0.34	3	79.91	33.47
4	True	0.08	0.64	5	35.05	39.52
5	True	0.07	0.34	3	60.43	23.62
6	True	0.08	0.56	3	47.37	25.22
7	True	0.15	0.33	3	73.59	24.25
8	False	0.59	ε	0	0	100
9	True	0.33	0.25	4	53.07	31.76
10	False	0.48	ε	0	0	100
11	False	0.63	ε	0	0	100
12	False	0.51	ε	0	0	100
13	True	0.12	0.51	3	37.38	27.21
14	True	0.17	0.6	7	35.98	25.74
15	True	0.24	0.42	2	27.63	44.99
16	False	0.51	ε	0	0	100
17	True	0.11	0.34	3	50.89	29.25
18	False	0.59	ε	0	0	100
19	True	0.06	0.39	4	48.21	21.67
20	True	0.14	0.4	2	21.86	44.17

Table A.18: Local Search results with Matrix, p-dP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	0.75	0.59	4	96.41	11.16
2	False	2.11	ε	0	0	100
3	True	0.32	0.33	4	41.63	15.05
4	True	0.23	0.43	5	27.22	10.92
5	True	2.05	0.5	4	40.1	8.02
6	True	0.4	0.69	5	80.41	6.75
7	True	0.32	0.39	4	36.99	9.57
8	False	1.58	ε	0	0	100
9	True	0.35	0.7	6	38.8	5.58
10	False	2.13	ε	0	0	100
11	False	1.47	ε	0	0	100
12	True	0.46	0.59	6	36.82	12.7
13	True	0.66	0.52	4	45.92	14.54
14	True	0.53	0.67	5	58.28	11.63
15	True	0.47	0.73	6	76.06	7.33
16	True	0.39	0.5	4	62.88	7.1
17	True	0.58	0.64	6	81.71	8.56
18	True	0.45	0.62	5	53.05	7.88
19	True	0.52	0.49	4	58.16	16.37
20	True	2.58	0.56	5	68.62	5.88

Table A.19: Local Search results with Queue, p-dP and FF strategies for Size 1 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	False	0.43	ε	0	0	100
2	False	0.48	ε	0	0	100
3	True	0.22	0.55	4	96.7	11.58
4	True	0.18	0.52	4	68.19	10.47
5	True	0.23	0.56	5	84.28	8.73
6	True	0.14	0.66	4	113.09	5.49
7	True	0.08	0.49	3	75.21	8.77
8	True	0.22	0.53	5	64.51	9.25
9	True	0.13	0.54	4	96.25	8.02
10	False	0.72	ε	0	0	100
11	False	0.43	ε	0	0	100
12	True	0.18	0.64	5	74.78	8.05
13	False	0.54	ε	0	0	100
14	True	0.2	0.56	4	80.09	15.59
15	True	0.24	0.78	6	95.25	11.87
16	False	0.43	ε	0	0	100
17	True	0.1	0.88	6	109.43	10.19
18	True	0.21	0.57	5	89.44	6.86
19	True	0.09	0.74	5	103.97	16.75
20	False	0.51	ε	0	0	100

A.3.2 Instances of Size 2

Table A.20: Local Search results with Matrix, PLP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	5.73	7.92	5	77.64	13.51
2	True	10.97	10.47	5	114.95	11.36
3	True	8.86	8.09	5	71.05	14.36
4	True	5.35	8.73	5	65.89	13.11
5	True	9.84	7.51	4	72.25	9.8
6	True	7.57	9.99	5	88.57	10.78
7	True	4.74	6.67	5	53.97	12.44
8	True	4.99	6.72	6	42.42	10.86
9	True	9.38	8.44	4	91.65	11.02
10	True	10.77	9.5	5	81.07	11.99
11	True	8.79	8.58	6	98.37	8.43
12	True	8.12	6.65	4	69.29	14.16
13	True	7.5	13.07	5	116.47	13.08
14	True	4.46	7.2	5	65.05	10.73
15	True	3.03	5.16	4	18.67	9.75
16	True	6.45	7.25	4	67.63	10.06
17	True	4.27	6.94	5	38.03	8.25
18	True	5.2	7.51	5	57.29	11.23
19	True	5.19	8.57	5	64.51	10.23
20	True	5.7	8.14	4	79.64	8.6

Table A.21: Local Search results with Matrix, p-dP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	2.13	6.26	4	27.3	8.56
2	True	4.35	10.01	5	58.88	8.02
3	True	8.15	13.28	5	68.24	11.25
4	True	3.47	9.37	5	46.52	9.94
5	True	2.82	7.2	4	31.95	10.7
6	True	3.29	7.87	5	40.04	8.44
7	True	3.72	9.33	4	46.89	9.15
8	True	3.35	8.64	5	39.06	13.36
9	True	5.0	10.57	4	56.92	12.4
10	True	4.88	9.28	6	36.41	14.37
11	True	4.09	9.48	5	51.21	8.88
12	True	6.1	9.45	4	59.81	11.16
13	True	3.83	12.53	5	49.78	13.03
14	True	3.07	9.52	5	54.22	7.8
15	True	2.63	9.01	6	52.13	8.72
16	True	5.99	11.46	6	60.72	9.96
17	True	2.69	10.07	5	46.67	9.87
18	True	3.14	8.76	5	45.84	10.74
19	True	3.08	6.81	4	37.12	10.69
20	True	7.81	11.72	4	73.31	10.68

Table A.22: Local Search results with Queue, PLP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	10.3	5	140.11	9.79
2	True	ε	13.14	5	151.9	12.47
3	True	ε	9.91	4	130.8	8.99
4	True	ε	10.47	5	110.77	15.82
5	True	0.03	9.2	4	107.28	8.89
6	True	ε	10.75	6	134.83	11.82
7	True	ε	8.59	4	99.17	10.91
8	True	ε	8.64	6	78.09	10.7
9	True	ε	12.81	5	148.45	12.73
10	True	ε	12.03	5	118.41	13.04
11	True	ε	11.94	5	153.94	6.2
12	True	ε	8.19	4	92.86	14.6
13	True	ε	14.93	5	195.0	8.39
14	True	ε	10.67	5	105.73	10.92
15	True	ε	9.55	5	68.18	9.93
16	True	0.15	10.81	5	114.79	7.97
17	True	ε	11.34	5	95.16	8.38
18	True	ε	8.57	5	97.3	11.51
19	True	ε	13.04	5	108.97	12.26
20	True	ε	10.13	5	116.31	9.08

Table A.23: Local Search results with Queue, p-dP and BF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	12.59	4	104.29	8.87
2	True	0.1	15.86	5	119.76	10.18
3	True	ε	17.05	5	110.68	10.68
4	True	ε	17.2	6	122.46	8.55
5	True	ε	12.68	4	93.14	13.91
6	True	ε	14.55	5	116.24	9.14
7	True	0.02	16.05	4	115.43	9.6
8	True	ε	14.09	4	108.17	9.81
9	True	ε	16.26	5	119.09	12.1
10	True	ε	16.22	5	128.94	11.44
11	True	ε	14.26	4	123.93	8.47
12	True	ε	14.41	5	136.5	11.63
13	True	ε	19.25	5	137.3	12.16
14	True	ε	16.48	5	123.47	11.84
15	True	ε	15.17	5	147.18	9.98
16	True	ε	16.81	5	133.85	11.07
17	True	ε	15.81	5	122.46	8.68
18	True	ε	13.5	4	114.57	11.35
19	True	ε	15.51	5	124.4	11.24
20	True	ε	15.79	5	121.35	11.86

Table A.24: Local Search results with Matrix, PLP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	5.73	4.45	4	75.6	14.5
2	True	11.57	5.72	5	114.32	11.62
3	True	8.93	5.62	4	70.51	14.63
4	True	5.64	5.78	4	66.91	12.58
5	True	10.04	5.14	5	66.59	12.77
6	True	7.54	5.97	5	84.18	12.86
7	True	4.65	4.96	5	55.78	11.41
8	True	5.01	4.77	5	43.86	9.96
9	True	8.81	5.17	5	89.65	11.95
10	True	10.63	6.05	5	76.24	14.33
11	True	8.01	5.29	5	98.34	8.44
12	True	8.9	4.57	5	68.2	14.71
13	True	7.04	7.71	5	126.24	9.15
14	True	4.82	4.6	4	61.63	12.58
15	True	3.07	3.75	4	20.84	8.1
16	True	6.83	6.43	6	67.83	9.95
17	True	4.28	4.86	5	40.83	6.38
18	True	5.12	4.74	4	54.98	12.53
19	True	5.53	5.82	6	61.24	12.01
20	True	5.43	5.84	5	78.65	9.11

Table A.25: Local Search results with Matrix, p-dP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	2.15	4.3	5	26.75	8.95
2	True	4.35	6.15	5	53.53	11.11
3	True	8.03	7.6	5	63.85	13.57
4	True	3.48	5.59	6	45.72	10.43
5	True	2.81	4.28	5	28.34	13.14
6	True	3.27	4.2	4	36.84	10.53
7	True	3.77	6.29	6	48.22	8.33
8	True	3.37	4.68	5	40.54	12.44
9	True	5.02	6.49	6	65.82	7.43
10	True	4.83	5.25	5	37.42	13.73
11	True	4.08	6.15	6	55.38	6.38
12	True	6.07	5.0	4	57.21	12.61
13	True	3.85	6.77	5	50.4	12.67
14	True	3.06	5.27	5	53.78	8.07
15	True	2.62	5.96	7	53.34	8.0
16	True	6.03	7.67	6	60.25	10.22
17	True	2.7	5.57	5	46.39	10.04
18	True	3.15	5.44	5	46.36	10.42
19	True	3.04	4.33	4	35.84	11.52
20	True	7.82	7.59	6	72.14	11.28

Table A.26: Local Search results with Queue, PLP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	5.92	5	141.92	9.11
2	True	ε	6.78	7	152.93	12.12
3	True	ε	7.2	6	118.29	13.92
4	True	ε	7.4	5	121.3	11.61
5	True	0.03	5.32	4	109.84	7.77
6	True	ε	7.49	5	133.86	12.18
7	True	ε	5.34	6	100.44	10.34
8	True	ε	4.71	5	79.94	9.78
9	True	ε	6.61	6	160.4	8.53
10	True	ε	6.41	5	118.95	12.82
11	True	ε	7.21	5	150.7	7.4
12	True	ε	4.95	4	101.45	10.8
13	True	ε	8.48	4	188.72	10.33
14	True	ε	5.56	4	101.75	12.64
15	True	ε	5.93	5	70.52	8.68
16	True	0.3	6.8	5	110.28	9.91
17	True	ε	5.89	5	91.29	10.2
18	True	ε	6.19	6	105.77	7.71
19	True	ε	7.5	5	117.24	8.79
20	True	ε	6.75	6	114.88	9.68

Table A.27: Local Search results with Queue, p-dP and FF strategies for Size 2 instances

Instance	Repaired	Time Repair (s)	Time LS (s)	LS Cycles	Rel. Improvement LS %	Rel. Optimality Gap %
1	True	ε	6.89	4	104.38	8.83
2	True	0.1	7.8	4	122.31	9.14
3	True	ε	9.05	4	104.6	13.26
4	True	ε	8.96	6	120.51	9.35
5	True	ε	6.59	5	96.45	12.44
6	True	ε	7.66	5	118.16	8.33
7	True	0.02	7.9	5	113.77	10.3
8	True	ε	7.9	5	106.91	10.36
9	True	ε	6.92	5	125.39	9.57
10	True	ε	7.83	7	130.59	10.8
11	True	ε	7.16	4	123.11	8.81
12	True	ε	7.43	6	139.79	10.4
13	True	ε	9.28	5	140.84	10.85
14	True	ε	8.39	5	129.4	9.51
15	True	ε	8.14	5	148.46	9.51
16	True	ε	8.37	5	139.87	8.78
17	True	ε	7.61	6	124.48	7.85
18	True	ε	7.26	5	113.92	11.62
19	True	ε	7.78	5	125.24	10.9
20	True	ε	10.02	5	128.47	9.02

A.4 GRASP parameter tuning tables

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.1	0.1	25	6	7.896	11.21	3.35	4.696	11.45
0.1	0.1	50	5.43	6.829	8.71	7.04	7.518	7.93
0.1	0.1	75	5.75	7.024	9.31	9.94	11.239	12.25
0.1	0.1	100	4.79	6.03	8.22	14.16	14.88	15.72
0.1	0.3	25	4.42	6.815	8.37	3.34	3.969	4.73
0.1	0.3	50	5.01	6.03	7.49	6.55	7.288	8.04
0.1	0.3	75	5.12	5.78	6.96	10.27	11.129	11.95
0.1	0.3	100	5.16	6.147	7.27	12.82	14.183	15.71
0.1	0.5	25	4.2	6.096	8.32	3.51	3.947	4.28
0.1	0.5	50	5.54	6.375	7.19	6.36	7.305	8.04
0.1	0.5	75	5.21	5.747	7.1	10.49	11.306	12.05
0.1	0.5	100	4.7	5.771	7.52	12.87	14.155	15.83
0.1	0.7	25	5.28	6.468	8.47	3.58	4.072	4.4
0.1	0.7	50	4.69	5.985	6.62	6.76	7.582	8.23
0.1	0.7	75	4.63	5.889	7.41	10.78	11.374	12.2
0.1	0.7	100	4.47	5.956	7.81	13.65	14.947	15.96
0.3	0.1	25	5.97	8.066	10.75	3.79	4.019	4.47
0.3	0.1	50	5.31	7.519	10.19	7.12	7.689	8.14
0.3	0.1	75	5.87	7.121	8.55	10.66	10.998	11.28
0.3	0.1	100	5.5	6.791	8.78	14.04	14.512	14.99
0.3	0.3	25	5.61	7.668	8.9	3.37	3.956	4.47
0.3	0.3	50	5.84	7.037	9.19	7.09	7.695	8.48

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.3	0.3	75	4.73	6.515	7.86	10.54	11.158	12.28
0.3	0.3	100	4.52	6.055	6.76	13.73	14.758	16.31
0.3	0.5	25	5.69	7.018	7.72	3.75	4.158	4.5
0.3	0.5	50	5.97	6.936	8.81	7.37	7.73	8.12
0.3	0.5	75	5.62	6.843	7.63	10.11	11.197	12.28
0.3	0.5	100	5.54	6.586	7.7	13.03	14.686	15.7
0.3	0.7	25	5.1	6.985	8.68	3.7	4.089	4.51
0.3	0.7	50	5.31	6.603	8.15	7.26	7.825	8.49
0.3	0.7	75	4.63	6.255	7.68	10.78	11.531	12.74
0.3	0.7	100	4.48	6.11	7.42	14.04	14.573	15.28
0.5	0.1	25	5.59	8.195	9.55	3.73	4.054	4.36
0.5	0.1	50	4.79	7.193	8.55	6.74	7.606	8.36
0.5	0.1	75	5.36	7.154	8.57	10.22	11.051	11.75
0.5	0.1	100	4.57	7.051	9.15	13.73	14.573	15.2
0.5	0.3	25	5.64	7.158	9.1	3.3	3.996	4.53
0.5	0.3	50	5.69	6.89	8.16	7	7.57	8.28
0.5	0.3	75	4.67	6.726	8.44	10.14	10.903	12.39
0.5	0.3	100	4.25	6.099	7.24	13.22	14.566	18.5
0.5	0.5	25	5.28	7.282	9.04	3.7	4.13	4.57
0.5	0.5	50	4.65	6.99	9	6.53	7.522	8.02
0.5	0.5	75	5.21	6.745	7.62	10.31	11.273	12.66
0.5	0.5	100	5.7	6.464	8.08	14	14.592	15.28
0.5	0.7	25	5.75	7.308	9.04	3.62	4.096	4.53
0.5	0.7	50	4.16	5.862	8.19	7.12	7.775	8.45
0.5	0.7	75	5.33	6.564	7.82	9.96	11.178	12.19

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.5	0.7	100	4.81	6.34	8.44	13.57	14.87	15.71
0.7	0.1	25	5.05	7.44	10.09	3.68	4.072	4.56
0.7	0.1	50	5.29	7.455	9.23	6.76	7.598	8.06
0.7	0.1	75	5.63	7.264	9.04	10.02	11.089	12.33
0.7	0.1	100	5.17	6.342	7.76	13.35	14.3	15.9
0.7	0.3	25	5.82	7.294	9.92	3.49	3.93	4.37
0.7	0.3	50	4.84	6.401	8.29	6.74	7.573	8.19
0.7	0.3	75	5.16	6.595	7.79	9.8	11.004	11.89
0.7	0.3	100	5.51	6.382	7.21	13.77	14.398	15.89
0.7	0.5	25	5.97	7.397	9.19	3.85	4.32	5.28
0.7	0.5	50	5.23	6.451	8.54	6.96	7.39	7.92
0.7	0.5	75	5.27	6.446	8.59	10.4	11.25	12.21
0.7	0.5	100	5.46	6.69	8.12	13.65	14.416	15.63
0.7	0.7	25	5.41	7.697	10.78	3.44	4.185	4.72
0.7	0.7	50	4.75	6.782	8.54	7.1	7.699	8.19
0.7	0.7	75	5.2	6.49	7.82	9.92	11.224	12.19
0.7	0.7	100	5.11	6.081	7.31	13.6	14.95	16.36

Table A.28: GRASP fine-tuning results for Size 1 instances

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.1	0.1	25	6.17	7.53	8.76	37.26	41.36	46.98
0.1	0.1	50	6.6	7.67	9.77	63.95	71.56	79.46
0.1	0.1	75	6.27	7.55	9.44	96.33	106.04	118.2
0.1	0.1	100	5.47	7.24	9.05	131.78	142.35	157.37
0.1	0.3	25	7.19	8.33	9.15	31.48	36.86	40.45
0.1	0.3	50	6.45	7.74	8.78	64.92	72.85	83.03
0.1	0.3	75	6.43	7.33	7.99	94.64	106.66	124.66
0.1	0.3	100	6.39	7.34	8.5	120.39	140.84	157.77
0.1	0.5	25	6.6	8.12	9.46	32.81	38.5	43.95
0.1	0.5	50	6.22	7.43	8.43	65.44	75.54	86.64
0.1	0.5	75	6.8	7.86	8.98	101.89	113.15	127.44
0.1	0.5	100	6.41	7.5	8.95	130.38	147.69	165.84
0.1	0.7	25	6.69	7.95	9.26	36.96	41.03	45.56
0.1	0.7	50	6.72	7.86	8.65	72.18	80.47	91.86
0.1	0.7	75	7.18	7.74	8.19	107.41	119.37	138.98
0.1	0.7	100	6.42	7.36	8.42	142.22	156.53	175.53
0.3	0.1	25	7.86	8.79	10.93	33.02	37.56	41.24
0.3	0.1	50	6.82	8.15	9.49	67.69	74.65	80.05
0.3	0.1	75	5.97	7.73	10.15	100.14	110.55	120.57
0.3	0.1	100	6.59	7.26	8.44	132.24	143.76	154.68
0.3	0.3	25	6.82	7.91	10.39	33.83	38.13	42.63
0.3	0.3	50	6.36	8.11	9.91	69.59	74.87	81.13
0.3	0.3	75	6.18	7.66	8.46	97.61	111.18	125.26
0.3	0.3	100	6.66	7.51	8.91	129.13	147.61	165.13

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.3	0.5	25	7.74	8.32	9.11	35.48	40.51	46.51
0.3	0.5	50	7.45	8.51	9.81	70.67	79.19	90.1
0.3	0.5	75	7.18	8.26	9.21	104.26	116.91	128.22
0.3	0.5	100	6.61	7.97	8.69	136.05	152.9	172.34
0.3	0.7	25	8.04	8.74	9.47	38.87	42.52	47.25
0.3	0.7	50	7.13	8.68	10.06	76.55	82.98	92.63
0.3	0.7	75	6.58	7.69	8.85	111.01	124.56	139.04
0.3	0.7	100	6.17	7.4	8.7	150.06	164.71	187.28
0.5	0.1	25	7.35	8.69	10.69	33.52	38.08	40.38
0.5	0.1	50	6.67	7.82	9.97	68.5	75.69	81.67
0.5	0.1	75	6.59	7.75	9.22	103.33	113.05	118.93
0.5	0.1	100	5.74	7.58	9.86	135.77	147.85	155.75
0.5	0.3	25	6.77	8.3	9.29	36.01	38.93	44.05
0.5	0.3	50	6.24	8.15	9.42	70.85	76.95	82.05
0.5	0.3	75	6.7	7.94	9.33	105.21	113.98	125.98
0.5	0.3	100	6.79	7.72	8.69	139.64	149.68	165.18
0.5	0.5	25	7.57	8.73	9.76	36.87	40.75	45.75
0.5	0.5	50	6.75	8.14	9.43	74.11	80.22	87.49
0.5	0.5	75	7.37	7.94	9.4	110.98	119.89	131.17
0.5	0.5	100	5.87	7.8	8.69	145.64	157.69	178.4
0.5	0.7	25	7.4	8.65	9.85	37.52	43.5	49.58
0.5	0.7	50	4.88	7.66	8.87	78.07	86.77	97.81
0.5	0.7	75	6.6	8.02	9.21	117.06	127.2	144.39
0.5	0.7	100	6.53	7.71	8.39	151.83	167.12	184.81
0.7	0.1	25	7.28	8.55	11.08	36.07	39.18	41.55

Continued on next page

α_a	α_l	iters	Optimality Gap (%)			Time (s)		
			Min	Avg	Max	Min	Avg	Max
0.7	0.1	50	6.49	7.64	10.35	70.86	77.73	81.21
0.7	0.1	75	6.53	7.49	9.26	106.47	114.23	123.11
0.7	0.1	100	5.43	7.08	9.54	136.44	151.39	158.22
0.7	0.3	25	7.16	8.59	9.47	36.87	39.57	44.23
0.7	0.3	50	7.4	7.91	8.8	74.33	79.89	89.21
0.7	0.3	75	6.28	7.53	9.49	106.64	117.07	127.05
0.7	0.3	100	5.69	7.73	9.64	140.18	153.92	169.95
0.7	0.5	25	6.68	8.55	11.24	39.56	41.34	44.64
0.7	0.5	50	6.66	7.85	9.02	74.17	80.61	88.41
0.7	0.5	75	6.85	7.77	8.52	110.73	121.06	130.91
0.7	0.5	100	5.85	7.62	8.84	142.74	160.38	179.97
0.7	0.7	25	7.81	8.8	10.17	41.24	44.47	48.08
0.7	0.7	50	7.53	8.5	9.49	78.36	87.07	94.29
0.7	0.7	75	6.39	7.44	8.33	120.84	130.08	143.76
0.7	0.7	100	4.49	7.49	8.99	156.56	170.61	187.39

Table A.29: GRASP fine-tuning results for Size 2 instances

A.5 GRASP Statistical tests

A.5.1 Statistical testing of Thread Tuning for Size 1 instances

Threads	Mean Runtime (s)	Standard Deviation (s)
1	38.28	1.88
2	33.47	2.11
3	24.32	1.39
4	22.96	1.25
5	19.00	0.98
6	18.62	0.91
7	17.01	0.98
8	16.89	0.66
9	16.09	0.70
10	15.60	0.62
11	15.32	0.77
12	15.34	0.68
13	14.82	0.61
14	14.74	0.42
15	14.48	0.49
16	14.60	0.64

Table A.30: Thread performance comparison for Size 1 instances

Repeated Measures ANOVA Summary

Source	F Value	Num DF	Den DF	Pr $> F$
Threads	1695.6740	15.0000	285.0000	0.0000

Table A.31: Summary of Repeated Measures ANOVA

The table contains the following:

Source: Refers to the independent variable or factor being tested, which in this case is "Threads", F Value: Represents the test statistic calculated by the ANOVA, num DF: Indicates the degrees of freedom for the numerator, which is the number of levels of the independent variable minus 1, den DF: Represents the degrees of freedom for the denominator, which is the total number of observations minus the number of levels of the independent variable, Pr $> F$: Denotes the p-value associated with the F statistic, indicating the statistical significance of the results.

The repeated measures ANOVA tests if there are statistically significant differences in the means of the same subjects under different conditions (in this case, runtime under different thread counts). The results show that there is a statistically significant difference in runtimes across different thread counts.

Following a significant ANOVA result, the Tukey's HSD post-hoc test helps identify which specific group(s) differ from each other. Tukey's HSD (Honestly Significant Difference) post-hoc test is a multiple comparison test used to find which means among a set of means differ from the rest. It is commonly used after a significant ANOVA result to determine which groups are different. The test compares all possible pairs of means and controls the familywise error rate.

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
1	2	-4809.85	0.0	-5962.0556	-3657.6444	True
1	3	-13961.85	0.0	-15114.0556	-12809.6444	True
1	4	-15317.35	0.0	-16469.5556	-14165.1444	True
1	5	-19279.9	0.0	-20432.1056	-18127.6944	True
1	6	-19662.85	0.0	-20815.0556	-18510.6444	True
1	7	-21270.95	0.0	-22423.1556	-20118.7444	True
1	8	-21387.25	0.0	-22539.4556	-20235.0444	True
1	9	-22187.1	0.0	-23339.3056	-21034.8944	True
1	10	-22683.45	0.0	-23835.6556	-21531.2444	True
1	11	-22960.0	0.0	-24112.2056	-21807.7944	True
1	12	-22937.5	0.0	-24089.7056	-21785.2944	True
1	13	-23461.05	0.0	-24613.2556	-22308.8444	True
1	14	-23545.4	0.0	-24697.6056	-22393.1944	True
1	15	-23803.15	0.0	-24955.3556	-22650.9444	True
1	16	-23681.65	0.0	-24833.8556	-22529.4444	True
2	3	-9152.0	0.0	-10304.2056	-7999.7944	True
2	4	-10507.5	0.0	-11659.7056	-9355.2944	True
2	5	-14470.05	0.0	-15622.2556	-13317.8444	True
2	6	-14853.0	0.0	-16005.2056	-13700.7944	True
2	7	-16461.1	0.0	-17613.3056	-15308.8944	True
2	8	-16577.4	0.0	-17729.6056	-15425.1944	True
2	9	-17377.25	0.0	-18529.4556	-16225.0444	True
2	10	-17873.6	0.0	-19025.8056	-16721.3944	True
2	11	-18150.15	0.0	-19302.3556	-16997.9444	True
2	12	-18127.65	0.0	-19279.8556	-16975.4444	True
2	13	-18651.2	0.0	-19803.4056	-17498.9944	True
2	14	-18735.55	0.0	-19887.7556	-17583.3444	True
2	15	-18993.3	0.0	-20145.5056	-17841.0944	True
2	16	-18871.8	0.0	-20024.0056	-17719.5944	True
3	4	-1355.5	0.0061	-2507.7056	-203.2944	True
3	5	-5318.05	0.0	-6470.2556	-4165.8444	True
3	6	-5701.0	0.0	-6853.2056	-4548.7944	True
3	7	-7309.1	0.0	-8461.3056	-6156.8944	True
3	8	-7425.4	0.0	-8577.6056	-6273.1944	True
3	9	-8225.25	0.0	-9377.4556	-7073.0444	True
3	10	-8721.6	0.0	-9873.8056	-7569.3944	True
3	11	-8998.15	0.0	-10150.3556	-7845.9444	True
3	12	-8975.65	0.0	-10127.8556	-7823.4444	True
3	13	-9499.2	0.0	-10651.4056	-8346.9944	True
3	14	-9583.55	0.0	-10735.7556	-8431.3444	True
3	15	-9841.3	0.0	-10993.5056	-8689.0944	True
3	16	-9719.8	0.0	-10872.0056	-8567.5944	True

Table A.32: Tukey HSD test results for threads Size 1 Part 1

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
4	5	-3962.55	0.0	-5114.7556	-2810.3444	True
4	6	-4345.5	0.0	-5497.7056	-3193.2944	True
4	7	-5953.6	0.0	-7105.8056	-4801.3944	True
4	8	-6069.9	0.0	-7222.1056	-4917.6944	True
4	9	-6869.75	0.0	-8021.9556	-5717.5444	True
4	10	-7366.1	0.0	-8518.3056	-6213.8944	True
4	11	-7642.65	0.0	-8794.8556	-6490.4444	True
4	12	-7620.15	0.0	-8772.3556	-6467.9444	True
4	13	-8143.7	0.0	-9295.9056	-6991.4944	True
4	14	-8228.05	0.0	-9380.2556	-7075.8444	True
4	15	-8485.8	0.0	-9638.0056	-7333.5944	True
4	16	-8364.3	0.0	-9516.5056	-7212.0944	True
5	6	-382.95	0.9989	-1535.1556	769.2556	False
5	7	-1991.05	0.0	-3143.2556	-838.8444	True
5	8	-2107.35	0.0	-3259.5556	-955.1444	True
5	9	-2907.2	0.0	-4059.4056	-1754.9944	True
5	10	-3403.55	0.0	-4555.7556	-2251.3444	True
5	11	-3680.1	0.0	-4832.3056	-2527.8944	True
5	12	-3657.6	0.0	-4809.8056	-2505.3944	True
5	13	-4181.15	0.0	-5333.3556	-3028.9444	True
5	14	-4265.5	0.0	-5417.7056	-3113.2944	True
5	15	-4523.25	0.0	-5675.4556	-3371.0444	True
5	16	-4401.75	0.0	-5553.9556	-3249.5444	True
6	7	-1608.1	0.0003	-2760.3056	-455.8944	True
6	8	-1724.4	0.0	-2876.6056	-572.1944	True
6	9	-2524.25	0.0	-3676.4556	-1372.0444	True
6	10	-3020.6	0.0	-4172.8056	-1868.3944	True
6	11	-3297.15	0.0	-4449.3556	-2144.9444	True
6	12	-3274.65	0.0	-4426.8556	-2122.4444	True
6	13	-3798.2	0.0	-4950.4056	-2645.9944	True
6	14	-3882.55	0.0	-5034.7556	-2730.3444	True
6	15	-4140.3	0.0	-5292.5056	-2988.0944	True
6	16	-4018.8	0.0	-5171.0056	-2866.5944	True
7	8	-116.3	1.0	-1268.5056	1035.9056	False
7	9	-916.15	0.306	-2068.3556	236.0556	False
7	10	-1412.5	0.0031	-2564.7056	-260.2944	True
7	11	-1689.05	0.0001	-2841.2556	-536.8444	True
7	12	-1666.55	0.0001	-2818.7556	-514.3444	True
7	13	-2190.1	0.0	-3342.3056	-1037.8944	True
7	14	-2274.45	0.0	-3426.6556	-1122.2444	True
7	15	-2532.2	0.0	-3684.4056	-1379.9944	True
7	16	-2410.7	0.0	-3562.9056	-1258.4944	True

Table A.33: Tukey HSD test results for threads Size 1 Part 2

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
8	9	-799.85	0.5502	-1952.0556	352.3556	False
8	10	-1296.2	0.0117	-2448.4056	-143.9944	True
8	11	-1572.75	0.0004	-2724.9556	-420.5444	True
8	12	-1550.25	0.0005	-2702.4556	-398.0444	True
8	13	-2073.8	0.0	-3226.0056	-921.5944	True
8	14	-2158.15	0.0	-3310.3556	-1005.9444	True
8	15	-2415.9	0.0	-3568.1056	-1263.6944	True
8	16	-2294.4	0.0	-3446.6056	-1142.1944	True
9	10	-496.35	0.9832	-1648.5556	655.8556	False
9	11	-772.9	0.6109	-1925.1056	379.3056	False
9	12	-750.4	0.6606	-1902.6056	401.8056	False
9	13	-1273.95	0.0149	-2426.1556	-121.7444	True
9	14	-1358.3	0.0059	-2510.5056	-206.0944	True
9	15	-1616.05	0.0002	-2768.2556	-463.8444	True
9	16	-1494.55	0.0011	-2646.7556	-342.3444	True
10	11	-276.55	1.0	-1428.7556	875.6556	False
10	12	-254.05	1.0	-1406.2556	898.1556	False
10	13	-777.6	0.6003	-1929.8056	374.6056	False
10	14	-861.95	0.4133	-2014.1556	290.2556	False
10	15	-1119.7	0.0671	-2271.9056	32.5056	False
10	16	-998.2	0.1781	-2150.4056	154.0056	False
11	12	22.5	1.0	-1129.7056	1174.7056	False
11	13	-501.05	0.9816	-1653.2556	651.1556	False
11	14	-585.4	0.9297	-1737.6056	566.8056	False
11	15	-843.15	0.4537	-1995.3556	309.0556	False
11	16	-721.65	0.7214	-1873.8556	430.5556	False
12	13	-523.55	0.9725	-1675.7556	628.6556	False
12	14	-607.9	0.9064	-1760.1056	544.3056	False
12	15	-865.65	0.4055	-2017.8556	286.5556	False
12	16	-744.15	0.6741	-1896.3556	408.0556	False
13	14	-84.35	1.0	-1236.5556	1067.8556	False
13	15	-342.1	0.9997	-1494.3056	810.1056	False
13	16	-220.6	1.0	-1372.8056	931.6056	False
14	15	-257.75	1.0	-1409.9556	894.4556	False
14	16	-136.25	1.0	-1288.4556	1015.9556	False
15	16	121.5	1.0	-1030.7056	1273.7056	False

Table A.34: Tukey HSD test results for threads Size 1 Part 3

The results of the Tukey Honestly Significant Difference (HSD) test reveal significant differences in the mean values of the dependent variable across different groups (Groups 1 to 16). Specifically:

There is a statistically significant difference ($p < 0.05$) in the mean values of the dependent variable between Group 1 and all other groups (Groups 2 to 16). Significant differences exist among most pairs of groups, with a few exceptions: a) There is no statistically significant difference between Group 5 and Group 6 ($p = 0.9989$). b) Group 7 does not significantly differ from Group 8 ($p = 1.0$) or Group 9 ($p = 0.306$). c) Group 8 does not significantly differ from Group 9 ($p = 0.5502$). d) Groups 9, 10, 11, and 12 do not significantly differ from each other (all $p > 0.05$). e) Groups 10 through 16 show no significant differences among themselves (all $p > 0.05$). The largest mean differences are observed between Group 1 and the higher-numbered groups, with the difference increasing as the group number increases. From Group 10 onwards, the differences between consecutive groups become smaller and statistically insignificant.

These findings suggest that the number of threads significantly influences the mean values of the dependent variable, which in this case it is the runtime, with Group 1 showing significant differences compared to all other groups, while certain pairs of groups exhibit no discernible differences. These results provide valuable insights into the relationship between the number of threads and the dependent variable, contributing to a deeper understanding of the experimental conditions under investigation.

Even though on average, the runtime does go down as more threads are added, the improvement eventually plateaus.

Effect Sizes (Cohen's d compared to 1 thread): Effect size is a measure of the magnitude of the difference between groups. Here, Cohen's d is calculated to quantify the difference in runtimes between 1 thread and the other thread counts.

Comparison	**Cohen's d**
Group 1 vs. Group 2	2.40
Group 1 vs. Group 3	8.44
Group 1 vs. Group 4	9.58
Group 1 vs. Group 5	12.82
Group 1 vs. Group 6	13.29
Group 1 vs. Group 7	14.15
Group 1 vs. Group 8	15.16
Group 1 vs. Group 9	15.62
Group 1 vs. Group 10	16.17
Group 1 vs. Group 11	15.95
Group 1 vs. Group 12	16.19
Group 1 vs. Group 13	16.75
Group 1 vs. Group 14	17.25
Group 1 vs. Group 15	17.28
Group 1 vs. Group 16	16.82

Table A.35: Cohen's d effect sizes for pairwise comparisons

These effect sizes indicate large differences between the mean values of the dependent variable for Group 1 compared to each of the other groups. The magnitude of these differences, as measured by Cohen's d, is substantial, suggesting a strong effect of the number of threads on the dependent variable.

Paired t-tests between adjacent thread counts: The paired t-tests are used to compare the means of the same group at two different times (in this case, runtimes between adjacent thread counts). The results provide insights into the significance of the performance differences between consecutive thread counts.

Pair	**T-Value**	**p-value**
1 vs. 2	15.07	5.07×10^{-12}
2 vs. 3	26.32	2.06×10^{-16}
3 vs. 4	6.29	4.89×10^{-6}
4 vs. 5	18.68	1.09×10^{-13}
5 vs. 6	1.99	0.0616
6 vs. 7	10.32	3.17×10^{-9}
7 vs. 8	0.49	0.6311
8 vs. 9	4.46	2.66×10^{-4}
9 vs. 10	4.01	7.48×10^{-4}
10 vs. 11	2.21	0.0399
11 vs. 12	-0.16	0.8779
12 vs. 13	3.68	0.0016
13 vs. 14	0.70	0.4947
14 vs. 15	2.68	0.0147
15 vs. 16	-0.78	0.4439

Table A.36: Paired t-tests results

Our paired t-tests indicate significant differences in mean values of the dependent variable between several pairs of groups. Specifically:

The mean values of Group 1 and Group 2 differ significantly ($t = 15.07, p < 0.001$). Significant differences are also observed between Group 2 and Group 3 ($t = 26.32, p < 0.001$), Group 3 and Group 4 ($t = 6.29, p < 0.001$), and Group 4 and Group 5 ($t = 18.68, p < 0.001$). The difference between Group 5 and Group 6 is not statistically significant at the conventional 0.05 level, but it is close ($t = 1.99, p = 0.0616$). Significant differences re-emerge between Group 6 and Group 7 ($t = 10.32, p < 0.001$). No significant difference is observed between Group 7 and

Group 8 ($t = 0.49, p = 0.6311$). Significant differences are found again between Group 8 and Group 9 ($t = 4.46, p < 0.001$), and Group 9 and Group 10 ($t = 4.01, p < 0.001$). The difference between Group 10 and Group 11 is significant at the 0.05 level ($t = 2.21, p = 0.0399$). No significant difference is found between Group 11 and Group 12 ($t = -0.16, p = 0.8779$). Significant differences are observed between Group 12 and Group 13 ($t = 3.68, p = 0.0016$). No significant differences are found between Group 13 and Group 14 ($t = 0.70, p = 0.4947$). A significant difference is observed between Group 14 and Group 15 ($t = 2.68, p = 0.0147$). Finally, no significant difference is found between Group 15 and Group 16 ($t = -0.78, p = 0.4439!$).

These findings suggest nuanced variations in the dependent variable across different groups, highlighting specific pairs where the differences are statistically significant. The pattern of significance indicates that:

The most substantial changes occur in the earlier groups (1 through 5), with highly significant differences between each pair. As the group numbers increase (representing an increase in the number of threads), the differences between adjacent groups become less consistently significant. There is a pattern of alternating significance and non-significance in the later groups, suggesting that performance improvements may plateau or become more incremental with higher thread counts. The lack of significant difference between the last two groups (15 and 16) might indicate that increasing the thread count beyond a certain point may not yield substantial performance improvements.

This pattern supports the hypothesis that as the number of threads increases, the difference between the values becomes less statistically significant, particularly in the higher thread count ranges. This could be due to factors such as diminishing returns from parallelization, potential overhead from thread management, or hardware limitations.

The one-way ANOVA test on the impact of threads on objective function

resulted in a $F-$statistic of approximately 0.1565 and a p-value of approximately 0.999. The high p-value suggests that there is no significant difference in the objective function value across different numbers of threads, meaning the number of threads does not have a statistically significant impact on the value based on this dataset.

In conclusion, increasing the number of threads has a statistically significant impact on the runtime, with no impact on the objective function value. Increasing from 1 to 10 threads and upwards shows no statistical difference, so increases beyond 10 may have diminishing returns.

A.5.2 Statistical testing of Thread Tuning for Size 2 instances

Threads	Mean Runtime (s)	Standard Deviation (s)
1	365.47	25.97
2	292.14	18.58
3	188.49	12.20
4	170.20	11.38
5	129.59	7.57
6	123.61	7.20
7	106.63	5.36
8	103.74	5.71
9	93.61	5.02
10	87.72	5.73
11	85.97	5.23
12	85.81	6.15
13	77.93	4.07
14	79.09	4.64
15	80.94	4.38
16	84.00	5.54

Table A.37: Thread performance comparison for Size 2 instances

Source	F Value	Num DF	Den DF	Pr > F
Threads	3086.2645	15.0000	285.0000	0.0000

Table A.38: Summary of Repeated Measures ANOVA

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
1	2	-73329.25	0.0	-84524.2403	-62134.2597	True
1	3	-176978.8	0.0	-188173.7903	-165783.8097	True
1	4	-195266.15	0.0	-206461.1403	-184071.1597	True
1	5	-235877.25	0.0	-247072.2403	-224682.2597	True
1	6	-241857.8	0.0	-253052.7903	-230662.8097	True
1	7	-258834.5	0.0	-270029.4903	-247639.5097	True
1	8	-261731.7	0.0	-272926.6903	-250536.7097	True
1	9	-271862.85	0.0	-283057.8403	-260667.8597	True
1	10	-277751.75	0.0	-288946.7403	-266556.7597	True
1	11	-279494.1	0.0	-290689.0903	-268299.1097	True
1	12	-279663.2	0.0	-290858.1903	-268468.2097	True
1	13	-287537.25	0.0	-298732.2403	-276342.2597	True
1	14	-286376.55	0.0	-297571.5403	-275181.5597	True
1	15	-284531.5	0.0	-295726.4903	-273336.5097	True
1	16	-281465.25	0.0	-292660.2403	-270270.2597	True
2	3	-103649.55	0.0	-114844.5403	-92454.5597	True
2	4	-121936.9	0.0	-133131.8903	-110741.9097	True
2	5	-162548.0	0.0	-173742.9903	-151353.0097	True
2	6	-168528.55	0.0	-179723.5403	-157333.5597	True
2	7	-185505.25	0.0	-196700.2403	-174310.2597	True
2	8	-188402.45	0.0	-199597.4403	-177207.4597	True
2	9	-198533.6	0.0	-209728.5903	-187338.6097	True
2	10	-204422.5	0.0	-215617.4903	-193227.5097	True
2	11	-206164.85	0.0	-217359.8403	-194969.8597	True
2	12	-206333.95	0.0	-217528.9403	-195138.9597	True
2	13	-214208.0	0.0	-225402.9903	-203013.0097	True
2	14	-213047.3	0.0	-224242.2903	-201852.3097	True
2	15	-211202.25	0.0	-222397.2403	-200007.2597	True
2	16	-208136.0	0.0	-219330.9903	-196941.0097	True
3	4	-18287.35	0.0	-29482.3403	-7092.3597	True
3	5	-58898.45	0.0	-70093.4403	-47703.4597	True
3	6	-64879.0	0.0	-76073.9903	-53684.0097	True
3	7	-81855.7	0.0	-93050.6903	-70660.7097	True
3	8	-84752.9	0.0	-95947.8903	-73557.9097	True
3	9	-94884.05	0.0	-106079.0403	-83689.0597	True
3	10	-100772.95	0.0	-111967.9403	-89577.9597	True
3	11	-102515.3	0.0	-113710.2903	-91320.3097	True
3	12	-102684.4	0.0	-113879.3903	-91489.4097	True
3	13	-110558.45	0.0	-121753.4403	-99363.4597	True
3	14	-109397.75	0.0	-120592.7403	-98202.7597	True
3	15	-107552.7	0.0	-118747.6903	-96357.7097	True
3	16	-104486.45	0.0	-115681.4403	-93291.4597	True

Table A.39: Tukey HSD test results for threads Size 2 Part 1

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
4	5	-40611.1	0.0	-51806.0903	-29416.1097	True
4	6	-46591.65	0.0	-57786.6403	-35396.6597	True
4	7	-63568.35	0.0	-74763.3403	-52373.3597	True
4	8	-66465.55	0.0	-77660.5403	-55270.5597	True
4	9	-76596.7	0.0	-87791.6903	-65401.7097	True
4	10	-82485.6	0.0	-93680.5903	-71290.6097	True
4	11	-84227.95	0.0	-95422.9403	-73032.9597	True
4	12	-84397.05	0.0	-95592.0403	-73202.0597	True
4	13	-92271.1	0.0	-103466.0903	-81076.1097	True
4	14	-91110.4	0.0	-102305.3903	-79915.4097	True
4	15	-89265.35	0.0	-100460.3403	-78070.3597	True
4	16	-86199.1	0.0	-97394.0903	-75004.1097	True
5	6	-5980.55	0.8974	-17175.5403	5214.4403	False
5	7	-22957.25	0.0	-34152.2403	-11762.2597	True
5	8	-25854.45	0.0	-37049.4403	-14659.4597	True
5	9	-35985.6	0.0	-47180.5903	-24790.6097	True
5	10	-41874.5	0.0	-53069.4903	-30679.5097	True
5	11	-43616.85	0.0	-54811.8403	-32421.8597	True
5	12	-43785.95	0.0	-54980.9403	-32590.9597	True
5	13	-51660.0	0.0	-62854.9903	-40465.0097	True
5	14	-50499.3	0.0	-61694.2903	-39304.3097	True
5	15	-48654.25	0.0	-59849.2403	-37459.2597	True
5	16	-45588.0	0.0	-56782.9903	-34393.0097	True
6	7	-16976.7	0.0	-28171.6903	-5781.7097	True
6	8	-19873.9	0.0	-31068.8903	-8678.9097	True
6	9	-30005.05	0.0	-41200.0403	-18810.0597	True
6	10	-35893.95	0.0	-47088.9403	-24698.9597	True
6	11	-37636.3	0.0	-48831.2903	-26441.3097	True
6	12	-37805.4	0.0	-49000.3903	-26610.4097	True
6	13	-45679.45	0.0	-56874.4403	-34484.4597	True
6	14	-44518.75	0.0	-55713.7403	-33323.7597	True
6	15	-42673.7	0.0	-53868.6903	-31478.7097	True
6	16	-39607.45	0.0	-50802.4403	-28412.4597	True
7	8	-2897.2	0.9999	-14092.1903	8297.7903	False
7	9	-13028.35	0.0072	-24223.3403	-1833.3597	True
7	10	-18917.25	0.0	-30112.2403	-7722.2597	True
7	11	-20659.6	0.0	-31854.5903	-9464.6097	True
7	12	-20828.7	0.0	-32023.6903	-9633.7097	True
7	13	-28702.75	0.0	-39897.7403	-17507.7597	True
7	14	-27542.05	0.0	-38737.0403	-16347.0597	True
7	15	-25697.0	0.0	-36891.9903	-14502.0097	True
7	16	-22630.75	0.0	-33825.7403	-11435.7597	True

Table A.40: Tukey HSD test results for threads Size 2 Part 2

Group 1	Group 2	Mean Difference	p-adj	Lower	Upper	Reject
8	9	-10131.15	0.1274	-21326.1403	1063.8403	False
8	10	-16020.05	0.0001	-27215.0403	-4825.0597	True
8	11	-17762.4	0.0	-28957.3903	-6567.4097	True
8	12	-17931.5	0.0	-29126.4903	-6736.5097	True
8	13	-25805.55	0.0	-37000.5403	-14610.5597	True
8	14	-24644.85	0.0	-35839.8403	-13449.8597	True
8	15	-22799.8	0.0	-33994.7903	-11604.8097	True
8	16	-19733.55	0.0	-30928.5403	-8538.5597	True
9	10	-5888.9	0.9084	-17083.8903	5306.0903	False
9	11	-7631.25	0.5828	-18826.2403	3563.7403	False
9	12	-7800.35	0.5435	-18995.3403	3394.6403	False
9	13	-15674.4	0.0002	-26869.3903	-4479.4097	True
9	14	-14513.7	0.0011	-25708.6903	-3318.7097	True
9	15	-12668.65	0.0108	-23863.6403	-1473.6597	True
9	16	-9602.4	0.1911	-20797.3903	1592.5903	False
10	11	-1742.35	1.0	-12937.3403	9452.6403	False
10	12	-1911.45	1.0	-13106.4403	9283.5403	False
10	13	-9785.5	0.1669	-20980.4903	1409.4903	False
10	14	-8624.8	0.3604	-19819.7903	2570.1903	False
10	15	-6779.75	0.7686	-17974.7403	4415.2403	False
10	16	-3713.5	0.9989	-14908.4903	7481.4903	False
11	12	-169.1	1.0	-11364.0903	11025.8903	False
11	13	-8043.15	0.4875	-19238.1403	3151.8403	False
11	14	-6882.45	0.7482	-18077.4403	4312.5403	False
11	15	-5037.4	0.9748	-16232.3903	6157.5903	False
11	16	-1971.15	1.0	-13166.1403	9223.8403	False
12	13	-7874.05	0.5264	-19069.0403	3320.9403	False
12	14	-6713.35	0.7814	-17908.3403	4481.6403	False
12	15	-4868.3	0.9816	-16063.2903	6326.6903	False
12	16	-1802.05	1.0	-12997.0403	9392.9403	False
13	14	1160.7	1.0	-10034.2903	12355.6903	False
13	15	3005.75	0.9999	-8189.2403	14200.7403	False
13	16	6072.0	0.8857	-5122.9903	17266.9903	False
14	15	1845.05	1.0	-9349.9403	13040.0403	False
14	16	4911.3	0.98	-6283.6903	16106.2903	False
15	16	3066.25	0.9999	-8128.7403	14261.2403	False

Table A.41: Tukey HSD test results for threads Size 2 Part 3

The results of the Tukey Honestly Significant Difference (HSD) test reveal significant differences in the mean runtime values across different thread groups (Groups 1 to 16). Specifically:

There is a statistically significant difference ($p < 0.05$) in the mean runtime between Group 1 (single thread) and all other groups (Groups 2 to 16). Significant differences exist among most pairs of groups, with some notable exceptions: a) There is no statistically significant difference between Group 5 and Group 6 ($p = 0.8974$). b) Group 7 does not significantly differ from Group 8 ($p = 0.9999$). c) Group 8 does not significantly differ from Group 9 ($p = 0.1274$). d) Groups 9, 10, 11, and 12 do not significantly differ from each other (all $p > 0.05$). e) Groups 10 through 16 show no significant differences among themselves (all $p > 0.05$). The largest mean differences are observed between Group 1 and the higher-numbered groups, with the difference increasing as the group number increases. The maximum mean difference is observed between Group 1 and Group 13 (-287,537.25). From Group 9 onwards, the differences between consecutive groups become smaller and statistically insignificant in most cases. The improvement in runtime is most pronounced in the early increases in thread count (from 1 to 8 threads), with statistically significant improvements observed at each step. After 9 threads, the improvements in runtime become less substantial and often not statistically significant, indicating a plateau in performance gains.

These findings suggest that the number of threads significantly influences the mean runtime, with single-threaded execution (Group 1) showing significant differences compared to all multi-threaded executions. The most substantial and statistically significant improvements occur when increasing threads from 1 to 9. Beyond 9 threads, while there are still some numerical improvements in runtime, these differences are often not statistically significant. This pattern indicates that there is a point of diminishing returns in adding more threads, likely due to factors such as increased overhead in thread management or limitations in the underlying hardware or workload parallelizability. The optimal thread count for this particular workload

appears to be around 9-13 threads, as further increases do not yield statistically significant improvements in runtime.

Cohen's d effect sizes:

Comparison	**Cohen's** d
Group 1 vs. Group 2	3.25
Group 1 vs. Group 3	8.72
Group 1 vs. Group 4	9.74
Group 1 vs. Group 5	12.33
Group 1 vs. Group 6	12.69
Group 1 vs. Group 7	13.80
Group 1 vs. Group 8	13.92
Group 1 vs. Group 9	14.53
Group 1 vs. Group 10	14.77
Group 1 vs. Group 11	14.92
Group 1 vs. Group 12	14.82
Group 1 vs. Group 13	15.47
Group 1 vs. Group 14	15.35
Group 1 vs. Group 15	15.28
Group 1 vs. Group 16	14.99

Table A.42: Cohen's d effect sizes for pairwise comparisons for Size 2

We get similar results as before, with the strongest effects when compared to running the single-threaded GRASP peaking around 13-14 threads.

Pair	**T-Value**	**p-value**
1 vs. 2	24.36	8.60×10^{-16}
2 vs. 3	44.61	1.07×10^{-20}
3 vs. 4	15.08	5.02×10^{-12}
4 vs. 5	33.05	2.97×10^{-18}
5 vs. 6	7.82	2.35×10^{-7}
6 vs. 7	17.68	2.97×10^{-13}
7 vs. 8	4.17	5.20×10^{-4}
8 vs. 9	18.67	1.11×10^{-13}
9 vs. 10	12.46	1.37×10^{-10}
10 vs. 11	2.96	8.12×10^{-3}
11 vs. 12	0.33	0.7447
12 vs. 13	12.16	2.07×10^{-10}
13 vs. 14	-2.68	0.0150
14 vs. 15	-3.75	1.36×10^{-3}
15 vs. 16	-3.29	3.89×10^{-3}

Table A.43: Paired t-tests results for Size 2

All pairs up to and including 9 vs. 10 show highly significant differences ($p < 0.001$), indicating substantial performance improvements with each increase in thread count up to 10 threads. The pair 10 vs. 11 shows a significant difference ($p < 0.01$), suggesting that there is still a measurable improvement when moving from 10 to 11 threads. The pair 11 vs. 12 shows no significant difference ($p = 0.7447$), indicating that the performance improvement plateaus at this point. Interestingly, the pair 12 vs. 13 shows a highly significant difference again ($p < 0.001$), suggesting a performance jump when moving to 13 threads. For pairs 13 vs. 14, 14 vs. 15, and 15 vs. 16, we see significant differences ($p < 0.05$), but with negative t-values. This suggests that performance actually decreases slightly with these additional threads.

This analysis suggests that the optimal thread count for this particular workload is likely 13-14, as it shows the last significant performance improvement before we start to see decreases. The slight performance decreases beyond 14 threads might be due to increased overhead from managing additional threads or resource contention.

The one-way ANOVA test on the impact of threads on objective function resulted in a $F-$statistic of approximately 0.1326 and a p-value of approximately 0.999. The high p-value suggests that there is no significant difference in the objective function value across different numbers of threads, meaning the number of threads does not have a statistically significant impact on the value based on this dataset, repeating the expected results from Size 1, so it is safe to say that using more threads has no impact, positive or negative, on the quality of the solution.

In conclusion, increasing the number of threads has a statistically significant impact on the runtime, with no impact on the objective function value. Increasing from 1 to 9 threads and upwards shows no statistical difference, and increases above 14 threads actually have negative performance, so 14 threads should be the ideal number to be used.

A.5.3 Statistical testing of Parameter Tuning for Size 1 instances

The One-Way ANOVA for the runtime of the GRASP Metaheuristic across different levels of iterations yielded an F-statistic of 8226.66 and a p-value of 0.0, indicating statistically significant differences in runtime across the iteration levels. For the α_a and α_l parameters, the runtime analysis resulted in F-statistics of 0.01 and 0.10, with p-values of 0.9985 and 0.9580 respectively, suggesting no significant impact on runtime. The analysis for the objective function value resulted in an F-statistic of 11.42 and a p-value of 2.652×10^{-7} for iterations, 4.83 and 0.0025 for α_a, and 7.09 and 1.089×10^{-4} for α_l, all showing statistically significant differences in solution

quality across these parameter levels.

Factorial ANOVA extends the concept of One-Way ANOVA by allowing for the examination of the effects of two or more independent variables (factors) simultaneously on a single dependent variable. It enables the analysis of not only the main effects of each factor but also the interaction effects between them. In the context of analyzing the performance of a metaheuristic algorithm, Factorial ANOVA allows us to understand not only how individual parameters (e.g., α_a, α_l and iterations) affect the algorithm's efficiency and solution quality but also how these parameters interact with each other to influence the outcomes. This comprehensive analysis is crucial for identifying optimal parameter settings, especially in complex systems where the interaction between parameters can significantly impact performance.

Effect	**Metric**	F**-Statistic**	p**-Value**
Time			
Main Effects - iters	Time	8347.26	< 0.000000
Main Effects - α_a	Time	0.43	0.735007
Main Effects - α_l	Time	4.18	0.006107
Interaction Effects (iters, α_a)	Time	0.85	0.572186
Interaction Effects (iters, α_l)	Time	0.94	0.493059
Interaction Effects (α_a, α_l)	Time	1.39	0.191076
Objective Value			
Main Effects - iters	Objective Value	11.64	< 0.0000002
Main Effects - α_a	Objective Value	5.07	0.001792
Main Effects - α_l	Objective Value	7.37	0.000075
Interaction Effects (iters, α_a)	Objective Value	0.14	0.998433
Interaction Effects (iters, α_l)	Objective Value	0.40	0.934506
Interaction Effects (α_a, α_l)	Objective Value	0.34	0.959751

Table A.44: Factorial ANOVA results for time and objective value for Size 1

The results show that the number of iterations had a very significant effect ($p < 0.000000$) on time, while α_a and α_l are not not significant. All interactions between parameters were not statistically significant. On the other hand, the number of iterations and both α parameters had a significant impact on objective values.

Detailed pairwise comparisons were conducted for both Time and Objective Function Value across different levels of iterations and α parameters identifying specific pairs that differ significantly. The Tukey HSD test results offer a comprehensive look at pairwise differences among the levels of iterations, highlighting significant disparities that inform the optimization of algorithm parameters.

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	3479.96	0.0	3298.29	3661.62	True
25	75	7076.15	0.0	6894.48	7257.82	True
25	100	10479.74	0.0	10298.08	10661.41	True
50	75	3596.19	0.0	3414.53	3777.86	True
50	100	6999.79	0.0	6818.12	7181.45	True
75	100	3403.59	0.0	3221.93	3585.26	True

Table A.45: Tukey HSD test results for iterations on runtime for Size 1

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	-3178.83	0.0045	-5611.92	-745.75	True
25	75	-3916.19	0.0002	-6349.28	-1483.11	True
25	100	-5336.96	0.0	-7770.04	-2903.87	True
50	75	-737.36	0.8632	-3170.45	1695.72	False
50	100	-2158.13	0.1026	-4591.21	274.96	False
75	100	-1420.76	0.4356	-3853.85	1012.32	False

Table A.46: Tukey HSD test results for iterations on objective value for Size 1

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	61.50	0.9991	-1084.63	1207.63	False
0.1	0.5	10.82	1.0	-1135.31	1156.95	False
0.1	0.7	-11.87	1.0	-1158.00	1134.26	False
0.3	0.5	-50.68	0.9995	-1196.81	1095.45	False
0.3	0.7	-73.37	0.9984	-1219.50	1072.76	False
0.5	0.7	-22.69	1.0	-1168.82	1123.44	False

Table A.47: Tukey HSD test results for α_a on runtime

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	3074.41	0.0077	604.60	5544.23	True
0.1	0.5	3046.51	0.0085	576.69	5516.32	True
0.1	0.7	2779.45	0.0202	309.63	5249.27	True
0.3	0.5	-27.91	1.0	-2497.72	2441.91	False
0.3	0.7	-294.96	0.9899	-2764.78	2174.85	False
0.5	0.7	-267.06	0.9925	-2736.87	2202.76	False

Table A.48: Tukey HSD test results for α_a on objective value for Size 1

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	-113.53	0.9942	-1259.40	1032.35	False
0.1	0.5	-32.23	0.9999	-1178.10	1113.65	False
0.1	0.7	129.78	0.9914	-1016.10	1275.65	False
0.3	0.5	81.30	0.9978	-1064.58	1227.18	False
0.3	0.7	243.30	0.9474	-902.58	1389.18	False
0.5	0.7	162.00	0.9835	-983.88	1307.88	False

Table A.49: Tukey HSD test results for α_l on runtime for Size 1

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	-3296.43	0.0033	-5753.47	-839.39	True
0.1	0.5	-3220.80	0.0043	-5677.84	-763.76	True
0.1	0.7	-4030.83	0.0002	-6487.86	-1573.79	True
0.3	0.5	75.63	0.9998	-2381.41	2532.67	False
0.3	0.7	-734.39	0.8680	-3191.43	1722.64	False
0.5	0.7	-810.03	0.8308	-3267.06	1647.01	False

Table A.50: Tukey HSD test results for α_l on objective value for Size 1

For the runtime of the algorithm, the test results show significant differences between all pairs of groups (p-adj = 0.0 for all comparisons). This indicates that the mean time is significantly different between each pair of groups tested. The magnitude of the mean difference (meandiff) increases as the difference between group numbers increases. For example, the difference between groups of 25 iterations and 50 iterations is smaller than the difference between groups of 25 iterations and 100 iterations. This suggests a trend where groups with higher iterations have significantly longer times. All comparisons were found to be significant, indicating that as the group number increases, so does the time, and these differences are not due to random chance.

Unlike the time, not all group comparisons for the objective function value are significant. Significant differences are observed in some pairs (e.g., 25 vs 50, 25 vs 75, 25 vs 100), while others show no significant difference. The significant differences do not follow a clear linear trend as observed with the time. This implies that the objective function value measure does not consistently increase or decrease with the group numbers in a manner that is statistically significant across all comparisons. The significant negative meandiff values (e.g., 25 vs 100, -5336.96) suggest that certain groups have significantly lower values compared to others. Specifically, group 25 tends to have higher value scores compared to other groups, indicating a decrease in value scores as the group number increases. However, when comparing 75 vs 100

iterations, the decrease is no longer statistically significant.

The time shows a clear and significant increase in values as the group number increases, suggesting a strong and consistent difference across all group comparisons.

The value, however, presents a more complex picture with significant differences only in certain group comparisons, indicating that while there are differences in value scores between specific groups, these differences are not uniformly observed across all groups.

α_a and α_l have no significant effect whatsoever on runtime, for any value they may take. Significant differences are observed between $\alpha_a = 0.1$ and all other values (0.3, 0.5, and 0.7). These comparisons have p-values < 0.05 and are rejecting the null hypothesis. The mean differences are positive, indicating that $\alpha_a = 0.1$ yields higher (worse) objective values compared to the other levels. There are no significant differences among values of 0.3, 0.5, and 0.7. Significant differences are observed between $\alpha_l = 0.1$ and all other values (0.3, 0.5, and 0.7). These comparisons have p-values < 0.05 and again reject the null hypothesis. The mean differences are negative, indicating that $\alpha_l = 0.1$ yields lower (better) objective values compared to the other levels. There are no significant differences among values of 0.3, 0.5, and 0.7.

These results suggest that while neither α_a nor α_l significantly affect runtime, they do impact the solution quality. Specifically, $\alpha_a = 0.1$ leads to worse solutions, while $\alpha_l = 0.1$ leads to better solutions compared to their respective higher values. For both parameters, values of 0.3, 0.5, and 0.7 perform similarly to each other in terms of solution quality.

Therefore, for instances of Size 1, the tuned parameters are set to the following values: iterations set to 100, although they increase time, they also improve objective function value (marginally when set to 100 iterations), but the time penalty is negligible; α_a set to 0.3 and α_l set to 0.1.

A.5.4 Statistical testing of Parameter Tuning for Size 2 instances

The One-Way ANOVA for the runtime of the GRASP Metaheuristic across different levels of iterations yielded an F-statistic of 5219.33 and a p-value of 0.0, indicating statistically significant differences in runtime across the iteration levels. For the α_a and α_l parameters, the runtime analysis resulted in F-statistics of 0.86 and 2.38, with p-values of 0.4637 and 0.0683 respectively, suggesting no significant impact on runtime. The analysis for the objective function value resulted in an F-statistic of 6.81 and a p-value of 1.602×10^{-4} for iterations, 1.46 and 0.2233 for α_a, and 0.51 and 0.6765 for α_l, showing statistically significant differences in solution quality across iteration levels, but not for the α parameters.

Effect	**Metric**	**F-Statistic**	**p-Value**
Time			
Main Effects - iters	Time	9562.34	< 0.000000
Main Effects - α_a	Time	40.01	1.364×10^{-23}
Main Effects - α_l	Time	110.64	4.445×10^{-57}
Interaction Effects (iters, α_a)	Time	3.76	1.281×10^{-4}
Interaction Effects (iters, α_l)	Time	7.91	4.337×10^{-11}
Interaction Effects (α_a, α_l)	Time	0.58	0.811264
Objective Value			
Main Effects - iters	Objective Value	6.58	0.000223
Main Effects - α_a	Objective Value	1.45	0.227379
Main Effects - α_l	Objective Value	0.51	0.678434
Interaction Effects (iters, α_a)	Objective Value	0.33	0.963511
Interaction Effects (iters, α_l)	Objective Value	0.15	0.998117
Interaction Effects (α_a, α_l)	Objective Value	0.12	0.999104

Table A.51: Factorial ANOVA results for time and objective value for Size 2

The results show that the number of iterations had a very significant effect ($p < 0.000000$) on time, while α_a and α_l also showed significant effects, contrary to the one-way ANOVA results. Interactions between iterations and both α parameters were statistically significant for time, but not between the α parameters themselves. For objective function values, only the number of iterations showed a significant impact, while neither the α parameters nor any interactions were significant. Detailed pairwise comparisons were conducted for both Time and Objective Function Value across different levels of iterations and α parameters identifying specific pairs that differ significantly. The Tukey HSD test results offer a comprehensive look at pairwise differences among the levels of iterations, highlighting significant disparities that inform the optimization of algorithm parameters.

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	38422.49	0.0	35962.50	40882.47	True
25	75	76418.65	0.0	73958.66	78878.64	True
25	100	113297.45	0.0	110837.46	115757.44	True
50	75	37996.16	0.0	35536.18	40456.15	True
50	100	74874.96	0.0	72414.98	77334.95	True
75	100	36878.80	0.0	34418.81	39338.79	True

Table A.52: Tukey HSD test results for iterations on runtime for Size 2

Group 1 (Iterations)	Group 2 (Iterations)	meandiff	p-adj	Lower	Upper	Reject
25	50	-4828.01	0.1775	-10949.99	1293.98	False
25	75	-7791.22	0.0061	-13913.20	-1669.24	True
25	100	-10179.63	0.0001	-16301.61	-4057.64	True
50	75	-2963.21	0.5973	-9085.20	3158.77	False
50	100	-5351.62	0.1107	-11473.60	770.36	False
75	100	-2388.41	0.7466	-8510.39	3733.58	False

Table A.53: Tukey HSD test results for iterations on objective value for Size 2

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	3237.18	0.908	-9189.15	15663.51	False
0.1	0.5	5409.20	0.6765	-7017.13	17835.53	False
0.1	0.7	7363.28	0.4223	-5063.05	19789.61	False
0.3	0.5	2172.02	0.9696	-10254.31	14598.35	False
0.3	0.7	4126.10	0.8278	-8300.23	16552.43	False
0.5	0.7	1954.08	0.9775	-10472.25	14380.41	False

Table A.54: Tukey HSD test results for α_a on runtime for Size 2

Group 1 (α_a values)	Group 2 (α_a values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	4385.91	0.2635	-1812.25	10584.06	False
0.1	0.5	4347.86	0.271	-1850.29	10546.02	False
0.1	0.7	2908.76	0.6215	-3289.40	9106.91	False
0.3	0.5	-38.04	1.0	-6236.20	6160.11	False
0.3	0.7	-1477.15	0.9277	-7675.30	4721.00	False
0.5	0.7	-1439.11	0.9326	-7637.26	4759.05	False

Table A.55: Tukey HSD test results for α_a on objective value for Size 2

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	872.45	0.9979	-11509.53	13254.43	False
0.1	0.5	5081.68	0.7157	-7300.30	17463.66	False
0.1	0.7	11496.06	0.0797	-885.92	23878.03	False
0.3	0.5	4209.23	0.8175	-8172.75	16591.21	False
0.3	0.7	10623.61	0.1216	-1758.37	23005.58	False
0.5	0.7	6414.38	0.5413	-5967.60	18796.35	False

Table A.56: Tukey HSD test results for α_l on runtime for Size 2

Group 1 (α_l values)	Group 2 (α_l values)	meandiff	p-adj	Lower	Upper	Reject
0.1	0.3	790.05	0.9879	-5422.02	7002.12	False
0.1	0.5	2650.46	0.6903	-3561.61	8862.53	False
0.1	0.7	2135.49	0.8124	-4076.58	8347.56	False
0.3	0.5	1860.41	0.8674	-4351.66	8072.48	False
0.3	0.7	1345.44	0.9444	-4866.63	7557.51	False
0.5	0.7	-514.98	0.9966	-6727.04	5697.09	False

Table A.57: Tukey HSD test results for α_l on objective value for Size 2

For the runtime of the algorithm, the test results show significant differences between all pairs of groups (p-adj = 0.0 for all comparisons). This indicates that the

mean time is significantly different between each pair of groups tested. The magnitude of the mean difference (meandiff) increases as the difference between group numbers increases. For example, the difference between groups of 25 iterations and 50 iterations is smaller than the difference between groups of 25 iterations and 100 iterations. This suggests a trend where groups with higher iterations have significantly longer times. All comparisons were found to be significant, indicating that as the group number increases, so does the time, and these differences are not due to random chance.

For the objective function value, significant differences are observed only in some pairs (25 vs 75, 25 vs 100), while others show no significant difference. The significant differences do not follow a clear linear trend as observed with the time. This implies that the objective function value measure does not consistently increase or decrease with the group numbers in a manner that is statistically significant across all comparisons. The significant negative meandiff values (e.g., 25 vs 100, -10179.63) suggest that certain groups have significantly lower values compared to others. Specifically, group 25 tends to have higher value scores compared to groups 75 and 100, indicating a decrease in value scores as the group number increases from 25 to 75 or 100. However, when comparing 50 vs 75, 50 vs 100, or 75 vs 100 iterations, the decreases are no longer statistically significant.

The time shows a clear and significant increase in values as the group number increases, suggesting a strong and consistent difference across all group comparisons. The value, however, presents a more complex picture with significant differences only in certain group comparisons, indicating that while there are differences in value scores between specific groups, these differences are not uniformly observed across all groups. Unlike in the previous results, α_a and α_l show no significant effect on either runtime or objective function value for any value they may take. All comparisons for both parameters across all levels have p-values > 0.05, failing to reject the null hypothesis. This suggests that changes in α_a and α_l do not lead to statistically significant changes in either runtime or solution quality.

These results suggest that while the number of iterations significantly affects both runtime and solution quality (to a certain extent), neither α_a nor α_l have a statistically significant impact on these metrics. This is in contrast to the factorial ANOVA results, which showed significant main effects for α_a and α_l on runtime. This discrepancy might be due to the different nature of the tests or potential interactions that are captured in the factorial ANOVA but not in the pairwise comparisons.

Therefore, for these instances of Size 2, the tuned parameters could be set as follows: iterations set to 75 or 100, as they improve objective function value significantly compared to 25 iterations, although they also increase time. The choice between 75 and 100 iterations would depend on the specific trade-off between solution quality and runtime that is acceptable for the problem at hand. As for α_a and α_l, given that they do not show statistically significant effects on either runtime or objective function value, their values could be set based on other considerations or left at default values, repeating the values for Size 1 of 0.3 and 0.1.

It is important to note that while the statistical tests do not show significant differences for α_a and α_l, there might still be practical differences that could be relevant in specific contexts. Additionally, the interaction effects observed in the factorial ANOVA for runtime suggest that the impact of these parameters might be more complex and intertwined with the number of iterations than the pairwise comparisons can reveal.

Future work might involve more detailed analysis of these interactions, or exploration of a wider range of values for α_a and α_l to see if significant effects emerge under different conditions. It might also be valuable to consider other performance metrics or to analyze the algorithm's behavior on different types of problem instances to get a more comprehensive understanding of the parameters' impacts.

A.6 Algorithms' graphical examples

A.6.1 The p-dispersion algorithm

Let us assume an instance of the problem with the configuration shown in the plot, with $|K| = 3$ to make the problem easier to follow. We have 3 center types along with their lower and upper bounds of centers to be used, figures and colors representing each type, along with 15 BUs, 14 centers and $p = 9$:

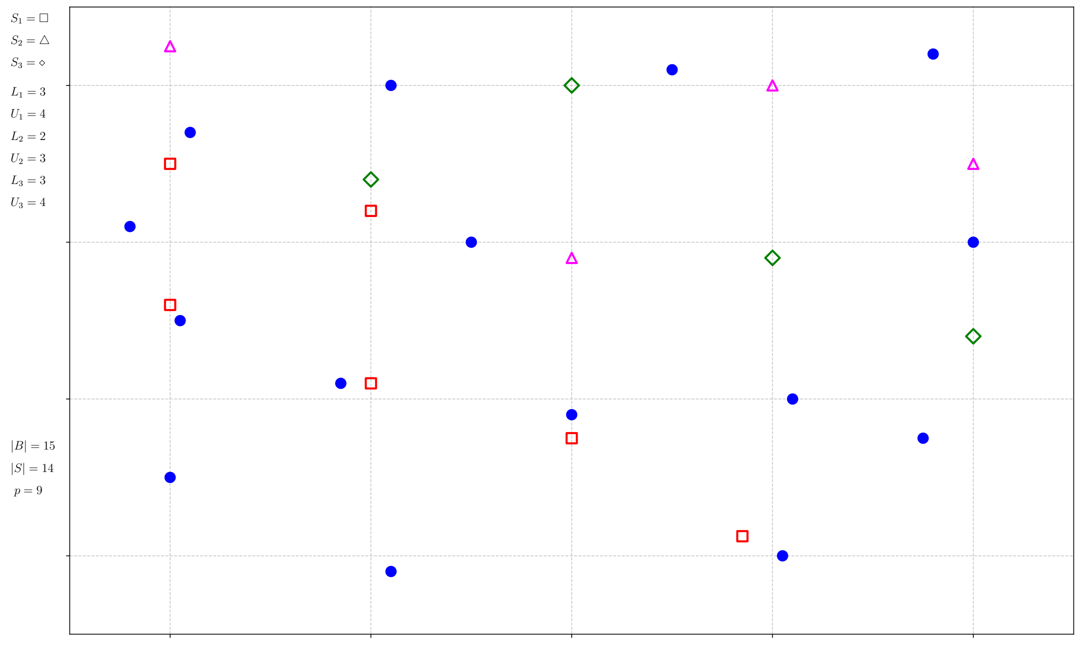


Figure A.1: Illustrative instance of the problem for p-dispersion algorithm

The location algorithm will solve $|K|$ p-dispersion subproblems with the heuristic presented in Algorithm 2. In this case, it will solve 3 subproblems, with the p of each problem set to the lower bound of the center type that it is trying to solve for, taking into account only the centers of that type, as follows:

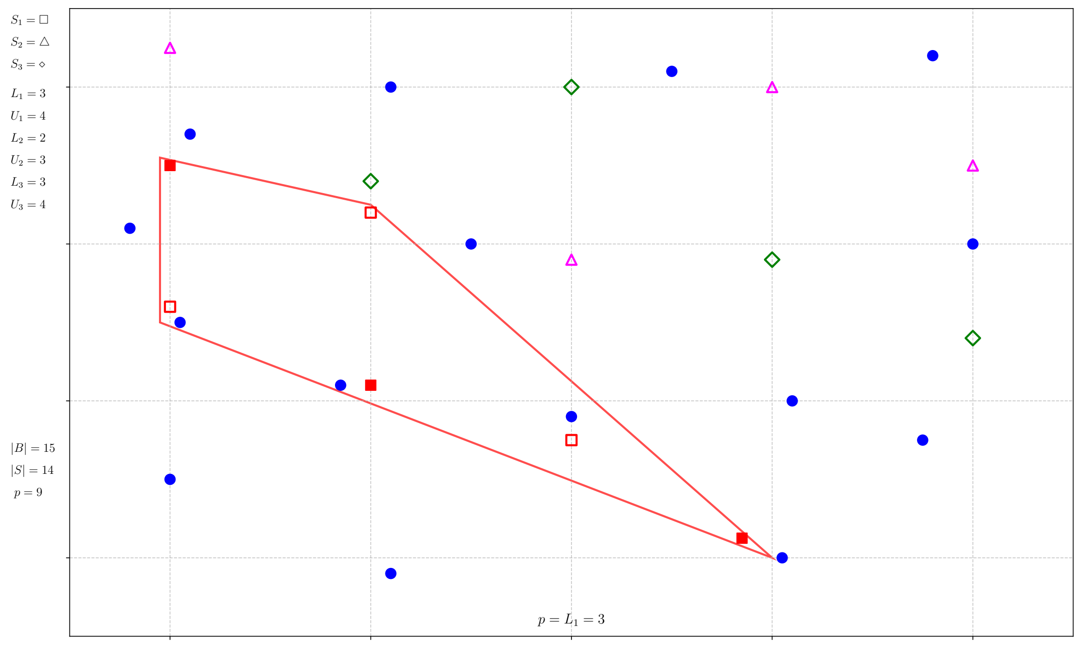


FIGURE A.2: Illustrative instance of the problem for p-dispersion for $k = 1$

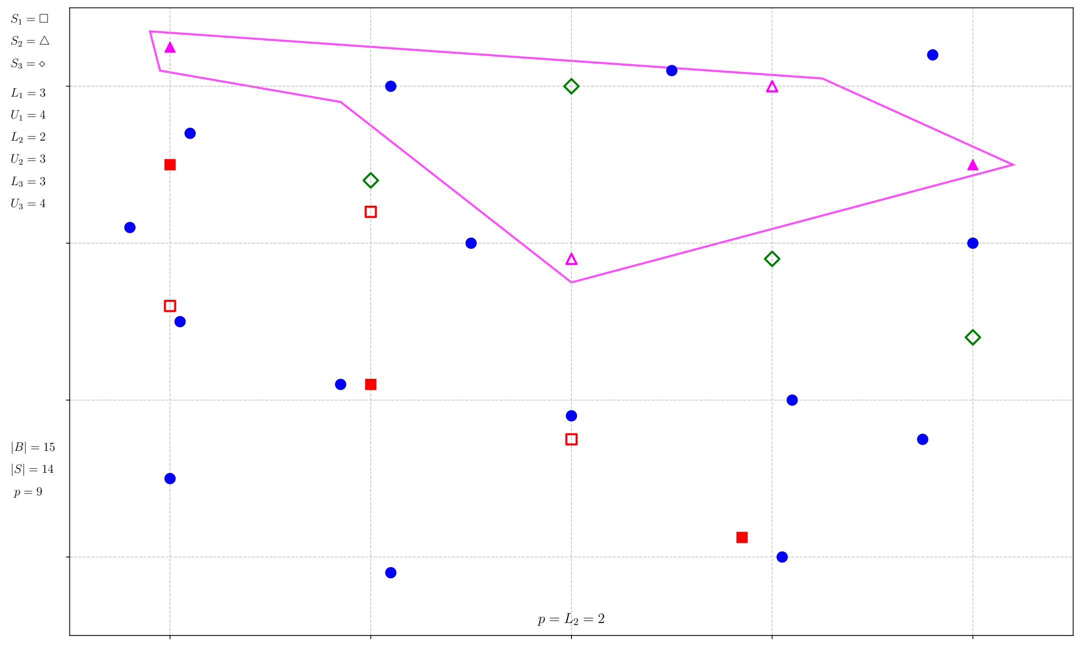


FIGURE A.3: Illustrative instance of the problem for p-dispersion for $k = 2$

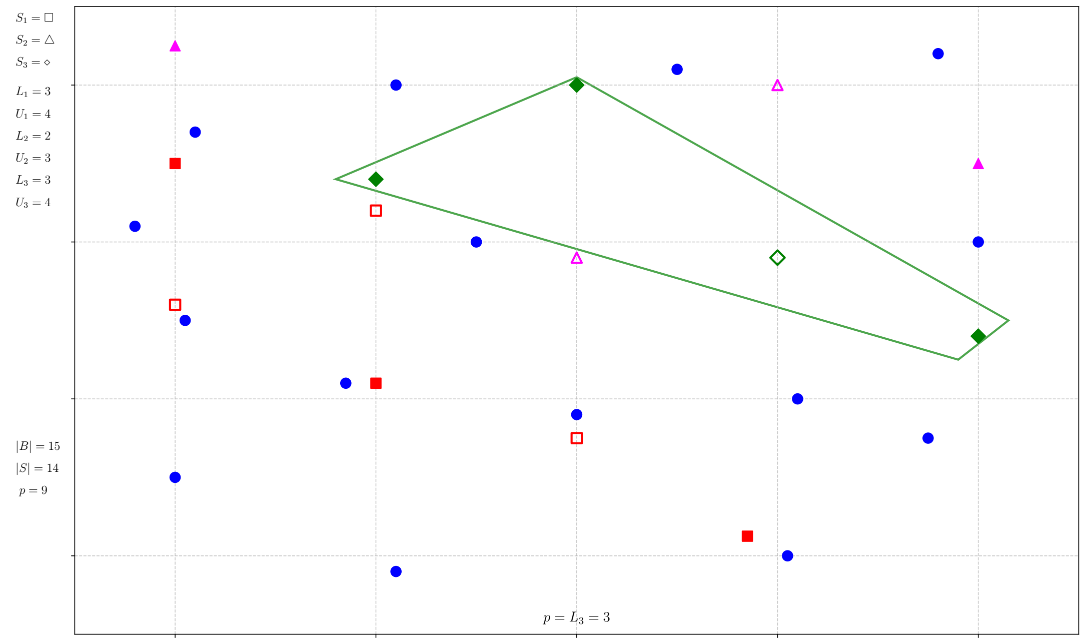


Figure A.4: Illustrative instance of the problem for p-dispersion for $k = 3$

After it is done with the subproblems, it combines the solutions into a larger set, and checks if the cardinality of the solution set is equal to the original p of the problem, which in this case is not, as it is missing one center. Therefore, it finally solves another p-dispersion problem, now taking all of the centers into account, adding centers to the solution only if their location does not produce an infeasible solution when taking into account U_k, in this case, it decides to add a new center of type 1, as shown in the following figures.

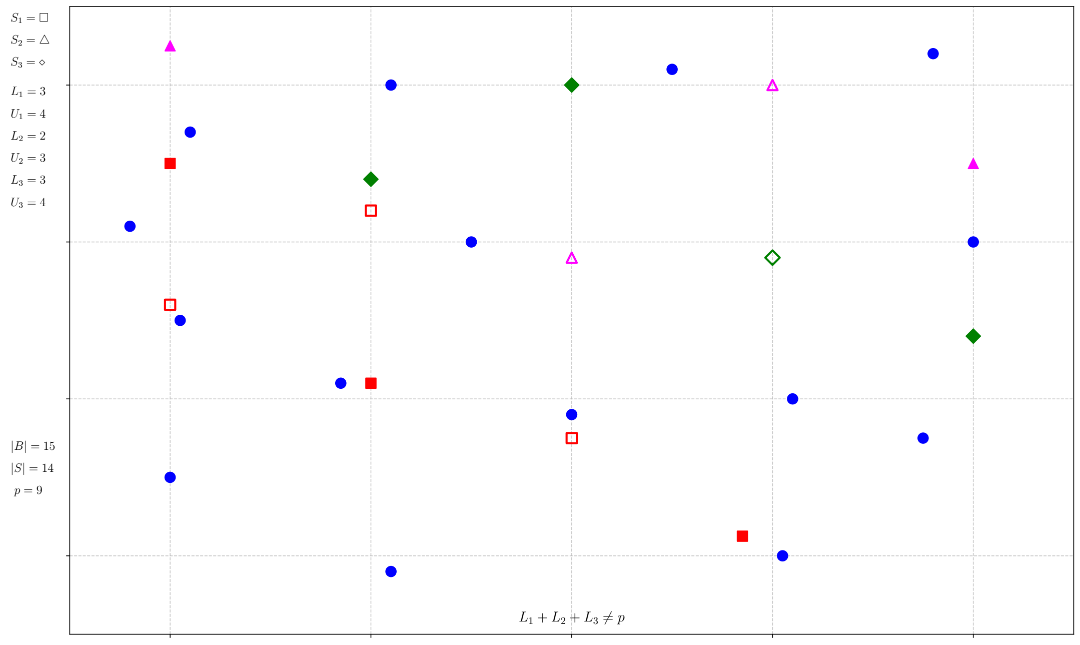


Figure A.5: Illustrative instance of the problem with the subproblems joined

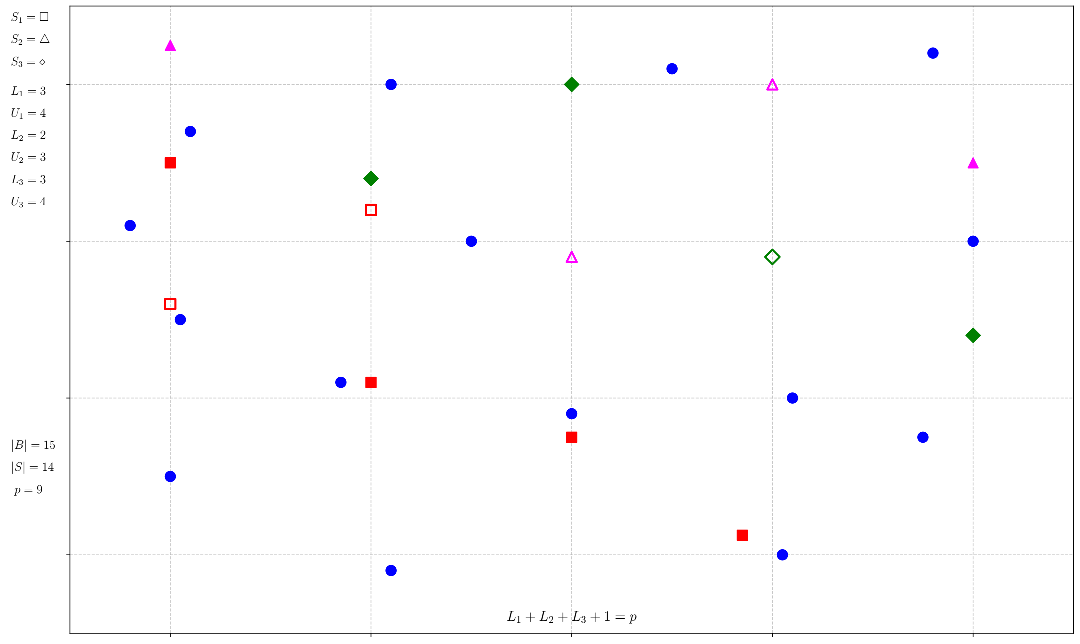


Figure A.6: Illustrative instance of the problem for p-dispersion as a whole

A.6.2 Allocation algorithm with opportunity cost

Let us assume an instance of the problem with 4 BUs and 2 centers, both opened, $Y_1 = 1, Y_2 = 1$, with the opportunity cost queue set as $\{1, 4, 2, 3\}$

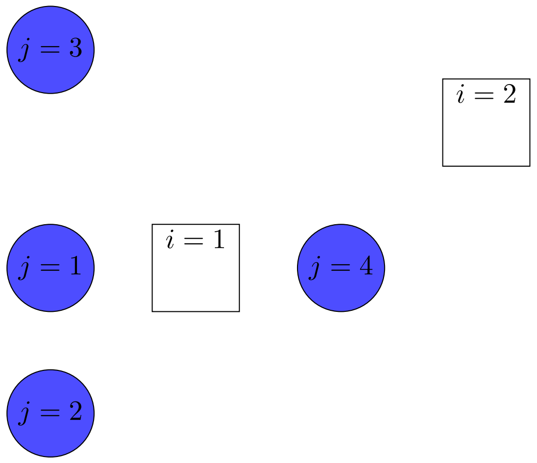


Figure A.7: Initial configuration

We dequeue the BU of $j = 1$ as it is the first element of the queue, and we assign it to its nearest center. We perform the calculation of the constraints of activity measures for $i = 1, j = 1$ and update the state of those constraints to the center, with $|M| = 3$. As a reminder, the description of the mathematical model is available in Section 3.3.

Let us assume the following values for $j = 1$: $v_1^1 = 10, v_1^2 = 30, v_1^3 = 23$ and the values of the constraints of the lower bounds for $i = 1$: $\mu_1^1 Y_1(1 - t^1) = 30, \mu_1^2 Y_1(1 - t^2) = 56, \mu_1^3 Y_1(1 - t^3) = 58$. We check if any constraint has reached the minimum value required, in this case, the center has not reached those values as seen in: $10 \leq 30$ for $m = 1$, $30 \leq 56$ for $m = 2$, $23 \leq 58$ for $m = 3$.

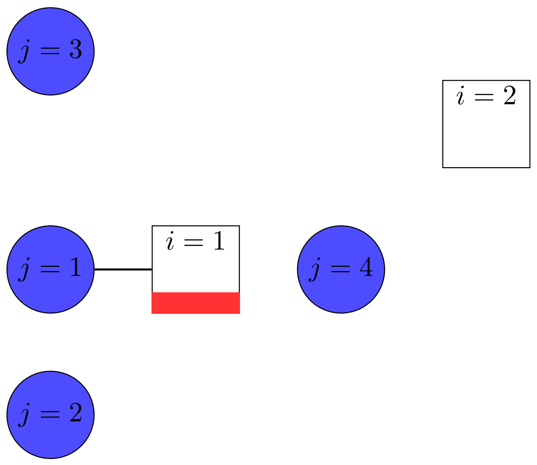


Figure A.8: First step with capacity constraints updated for $i = 1, j = 1$

We then dequeue the next BU, which is $j = 4$. The values of the activity measures that the BU provide are as follows: $j = 4$: $v_4^1 = 21, v_4^2 = 28, v_4^3 = 35$ and we update the values of $i = 1$: $10 + 21 > 30$ for $m = 1$, $30 + 28 > 56$ for $m = 2$, $23 + 35 > 58$ for $m = 3$. In this case, all activity measures have reached the minimum value required in the constraint, therefore we update the capacities of $i = 1$ and mark it as unavailable for future allocations.

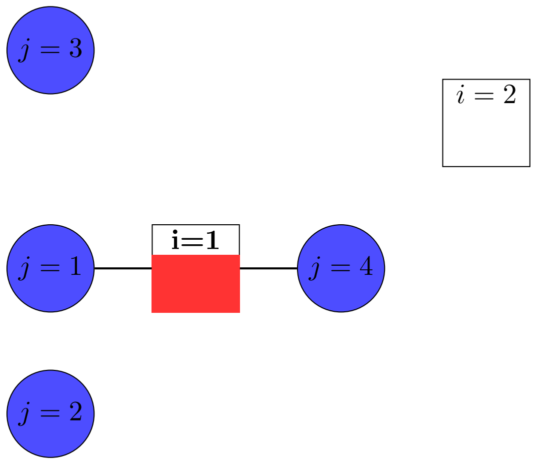


Figure A.9: Second step showing allocation from $j = 4$ to $i = 1$, unavailable.

We dequeue the next element, $j = 2$. In this case, its nearest center is $i = 1$, but it is unavailable to be allocated to, as it now complies with the constraints of the lower

bounds for the activity measures target, this forces other centers to be used, guiding the solution towards feasibility. As such, we allocate $j = 2$ to $i = 2$ and update the values for that center, with the values of the BU being: $v_2^1 = 15, v_2^2 = 37, v_2^3 = 25$, the constraints of the center being $\mu_1^1 Y_2(1-t^1) = 40, \mu_1^2 Y_2(1-t^2) = 50, \mu_1^3 Y_2(1-t^3) = 69$, with the state of the constraints being : $15 \leq 40$ for $m = 1$, $37 \leq 50$ for $m = 2$, $25 \leq 69$ for $m = 3$.

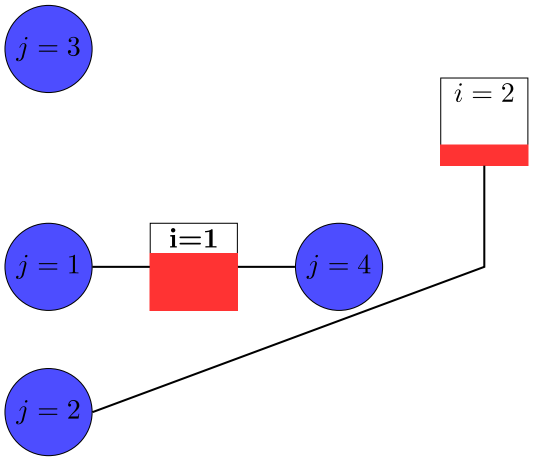


Figure A.10: Third step showing allocation from $j = 2$ to $i = 2$

Finally, we allocate $j = 3$ to $i = 2$, as it is the last element in the queue. We update the center capacities with the values from the BU $j = 3$: $v_3^1 = 26, v_3^2 = 30, v_3^3 = 48$, with the state of the constraints being : $15 + 26 > 40$ for $m = 1$, $37 + 30 > 50$ for $m = 2$, $25 + 48 > 69$ for $m = 3$. This makes $i = 2$ unavailable for future centers.

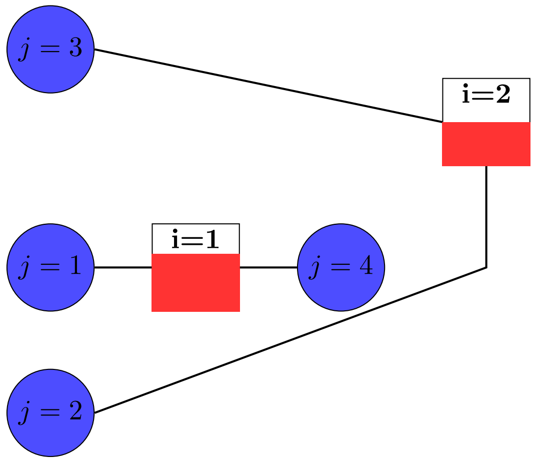


Figure A.11: Fourth step showing allocation from $j = 3$ to $i = 2$

The final assumption is that we introduce a fifth BU, $j = 5$, to illustrate what would happen in the case that all centers are unavailable due to their activity measures constraints.

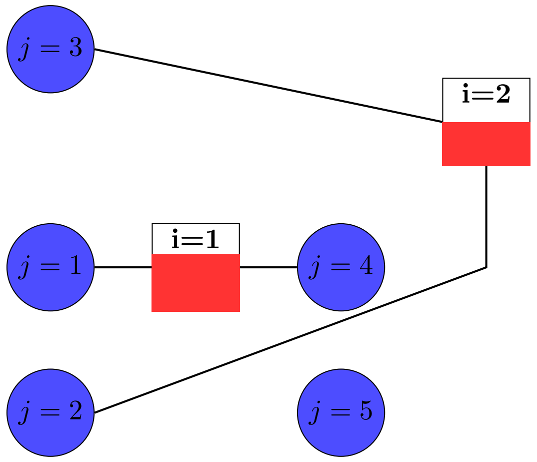


Figure A.12: Fifth step introducing $j = 5$

To handle unassigned BUs, we then switch to the risk threshold as the criterion of allocation, assigning the 5th BU to the center with the highest residual capacity in the risk threshold. By calculating the current used capacity in the risk threshold for each center, summing the values of the risks provided by each BU, we obtain the following results: we sum the risk values for $j = 1, j = 4$ as they are assigned

to $i = 1$, with a risk threshold of $\beta_1 = 120$, so the residual capacity for $i = 1$ is $r_1 = 40, r_4 = 15$, $120 - (40 + 15) = 65$, for $i = 2$ with a risk threshold of $\beta_2 = 150$ we find that the assignments are $j = 2$ and $j = 3$, $r_2 = 30, r_3 = 60$, then the capacity is $150 - (30 + 60) = 60$. In this case, $i = 1$ has the highest residual capacity in the risk threshold, so we assign $j = 5$ to it, and perform one last update on the state of the constraints, as shown in the increase of the red square representing the capacities used.

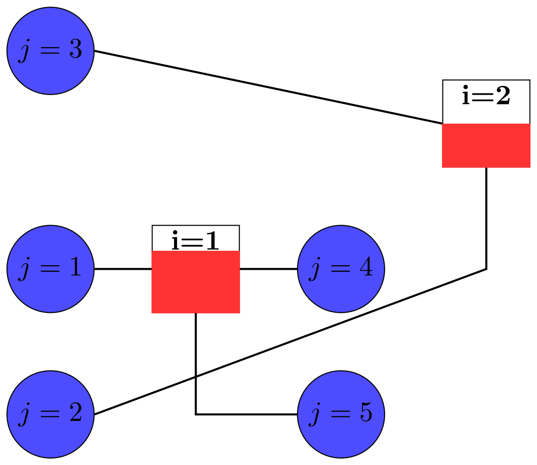


FIGURE A.13: Sixth step showing allocation from $j = 5$ to $i = 1$

Bibliography

G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483–485, New York, USA, 1967. Association for Computing Machinery.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2014.

G. Bruton, S. Khavul, D. Siegel, and M. Wright. New financial alternatives in seeding entrepreneurship: Microfinance, crowdfunding, and peer–to–peer innovations. *Entrepreneurship Theory and Practice*, 39(1):9–26, 2015.

J. Cano-Belmán, R. Z. Ríos-Mercado, and M. A. Salazar-Aguilar. Commercial territory design for a distribution firm with new constructive and destructive heuristics. *International Journal of Computational Intelligence Systems*, 5(1):126–147, 2012.

I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

E. Erkut and S. Neuman. Comparison of four models for dispersing facilities. *INFOR: Information Systems and Operational Research*, 29(2):68–86, 1991.

E. Erkut, Y. Ülküsal, and O. Yeniçerioğlu. A comparison of p-dispersion heuristics. *Computers & Operations Research*, 21(10):1103–1113, 1994.

T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

E. Fernández, J. Kalcsics, S. Nickel, and R. Z. Ríos-Mercado. A novel maximum dispersion territory design model arising in the implementation of the WEEE-directive. *Journal of the Operational Research Society*, 61(3):503–514, 2010.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL : `https://www.gurobi.com`.

P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

S. W. Hess, J. B. Weaver, H. J. Siegfeldt, J. N. Whelan, and P. A. Zitlau. Nonpartisan political redistricting by computer. *Operations Research*, 13(6):998–1006, 1965.

J. Kalcsics. Towards a unified territorial design approach - applications, algorithms and gis integration. *TOP*, 13:1–56, 2005.

J. Kalcsics and R. Z. Ríos-Mercado. Districting problems. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, chapter 25, pages 703–741. Springer, Cham, Switzerland, 2nd edition, 2019.

J. Kallrath. *Modeling Languages in Mathematical Optimization*. Springer, New York, USA, 2004.

R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

S. Khandker. Microfinance and poverty: Evidence using panel data from Bangladesh. *World Bank Economic Review*, 19:263–286, 02 2005.

S. Koranne. Hierarchical data format 5: Hdf5. In *Handbook of Open Source Tools*, pages 191–200. Springer, New York, USA, 2011.

G. Laporte, S. Nickel, and F. Saldanha Da Gama, editors. *Location Science*. Springer, Cham, Switzerland, 2019.

J. F. López, T. Ekin, F. Mendez Mediavilla, and J. A. Jimenez. Hybrid heuristic for dynamic location-allocation on micro-credit territory design. *Computacion y Sistemas*, 19(4):783–804, 2015.

J. F. López, T. Ekin, F. Mendez Mediavilla, and J. A. Jimenez. Risk-balanced territory design optimization for a micro finance institution. *Journal of Industrial & Management Optimization*, 16(2):741–758, 2020.

F. A. Medrano. The complete vertex p-center problem. *EURO Journal on Computational Optimization*, 8(3-4):327–343, 2020.

M. S. R. Monteiro. Bank-branch location and sizing under economies of scale. Master's thesis, Universidade do Porto, Portugal, 2005.

M. S. R. Monteiro and D. B. M. M. Fontes. Locating and sizing bank-branches by opening, closing or maintaining facilities. In H.-D. Haasis, H. Kopfer, and J. Schönberger, editors, *Operations Research Proceedings 2005*, pages 303–308, Berlin, Germany, 2006. Springer.

D. R. Quevedo-Orozco and R. Z. Ríos-Mercado. Improving the quality of heuristic solutions for the capacitated vertex p-center problem through iterated greedy local search and variable neighborhood descent. *Computers & Operations Research*, 62: 133–144, 2015.

S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

R. Z. Ríos-Mercado and J. F. Bard. An exact algorithm for designing optimal districts in the collection of waste electric and electronic equipment through an improved reformulation. *European Journal of Operational Research*, 276(1):259–271, 2019.

R. Z. Ríos-Mercado, editor. *Optimal Districting and Territory Design*, volume 284 of *International Series in Operations Research and Management Science*. Springer, Cham, Switzerland, 2020.

R. Z. Ríos-Mercado and H. J. Escalante. GRASP with path relinking for commercial districting. *Expert Systems with Applications*, 44:102–113, 2016.

R. Z. Ríos-Mercado and E. Fernández. A reactive GRASP for a commercial territory design problem with multiple balancing requirements. *Computers & Operations Research*, 36(3):755–776, 2009.

R. Z. Ríos-Mercado and J. F. López-Pérez. Commercial territory design planning with realignment and disjoint assignment requirements. *Omega*, 41(3):525–535, 2013.

R. Z. Ríos-Mercado, A. M. Álvarez Socarrás, A. Castrillón, and M. C. López-Locés. A location-allocation-improvement heuristic for districting with multiple-activity balancing constraints and p-median-based dispersion minimization. *Computers & Operations Research*, 126:105106, 2021.

M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and M. Cabrera-Ríos. New models for commercial territory design. *Networks and Spatial Economics*, 11(3):487–507, 2011.

M. A. Salazar-Aguilar, R. Z. Ríos-Mercado, and J. L. González-Velarde. GRASP strategies for a bi-objective commercial territory design problem. *Journal of Heuristics*, 19(2):179–200, 2013.

M. G. Sandoval, J. A. Díaz, and R. Z. Ríos-Mercado. An improved exact algorithm for a territory design problem with p-center-based dispersion minimization. *Expert Systems with Applications*, 146:113150, 2020.

J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*. Wiley, New York, USA, 2003.

E. D. Taillard. *Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem.* Graduate Texts in Operations Research. Springer International Publishing, Cham, Switzerland, 2023.

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Acta de Titulación

Institución 190001
Programa 501363

642373/1847972/63753

52094
Acta Núm. ....

En la Ciudad de Monterrey, capital del Estado de Nuevo León, al día 25 del mes de Junio del año 2024, siendo las 12:00 horas, reunidos en las instalaciones de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA de la Universidad Autónoma de Nuevo León, los(as) profesores(as) DR. ROGER ZIRAHUEN RÍOS MERCADO, DRA. MARÍA ANGÉLICA SALAZAR AGUILAR, DR. VINCENT ANDRÉ LIONEL BOYER , quienes fueron designados por la dirección de la escuela o facultad para integrar el Comité de Titulación de EDUARDO SALAZAR TREVIÑO, quien cursó y aprobó todas y cada una de las unidades de aprendizaje del Programa Educativo de LICENCIATURA COMO INGENIERO EN TECNOLOGÍA DE SOFTWARE, tal como lo dispone la Ley Orgánica de la Universidad Autónoma de Nuevo León publicada en el Periódico Oficial el siete de junio de mil novecientos setenta y uno, y en su Título Quinto: De la titulación, Capítulo I, De la obtención del título, del Reglamento para la Admisión, Permanencia y Egreso de los Alumnos de la Universidad Autónoma de Nuevo León, aprobado el 8 de agosto de 2019 y en el Reglamento Interno de la Escuela o Facultad.

Se procedió a tomar la Protesta de Ley por el Presidente del Comité de Titulación y en cumplimiento de lo dispuesto por los preceptos legales y reglamentarios, firman la presente acta los profesores(as), ante la presencia del Secretario del Comité que da fe.

PRESIDENTE

[illegible]

DR. ROGER ZIRAHUEN RÍOS MERCADO

SECRETARIO

[illegible]

DRA. MARÍA ANGÉLICA SALAZAR AGUILAR

VOCAL

[illegible]

DR. VINCENT ANDRÉ LIONEL BOYER

El suscrito, Director de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA, CERTIFICA que las firmas que aparecen en la presente acta son auténticas y las mismas que utilizan los C.C. profesores mencionados en ella.

U.A.N.L.
DIRECCIÓN

[illegible]

DR. ARNULFO TREVIÑO CUBERO

El C. Secretario General de la Universidad Autónoma de Nuevo León, CERTIFICA que la firma del C. Director de la FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA que aparece en la presente acta, es auténtica.

[illegible]

DR. JUAN PAURA GARCÍA

SECRETARÍA GENERAL
DEPARTAMENTO ESCOLAR Y DE ARCHIVO

jmor/esb El recibir de conformidad este documento académico, compromete al interesado a dar el uso legal y correcto del mismo. Quien altere cualquiera de las partes que lo conforman, invalida inmediatamente su autenticidad, haciéndose acreedor a la aplicación de las sanciones previstas en las leyes y reglamentos de la UANL; independientemente de los efectos legales que procedan. IA 24/06/24

UANL-T
013744

ORIGINAL

LPT-23