

SELECTED TOPICS OF OPTIMIZATION **GENERAL ASSIGNMENT** **PROBLEM**

PROFESSOR: Roger Zirahuén Ríos Mercados

SEMESTER: JANUARY - JUNE 2024

Group: 001

Team B:

Alejandro Sebastian Carranza Rodriguez 1930968 ITS

Miguel Angel Ramirez Gutierrez 1926266 ITS



INTRODUCTION

The **Generalized Assignment Problem** (GAP) is an optimization challenge commonly encountered in real-world situations where resources must be allocated efficiently to tasks or jobs. It involves **assigning** a set of **resources**, each with specific **capacities**, to a set of **tasks**, each with different requirements and associated costs.

The main goal is to **minimize** the total cost of assigning resources to tasks while ensuring that each resource's capacity is not exceeded.



GAP ADVANTAGES

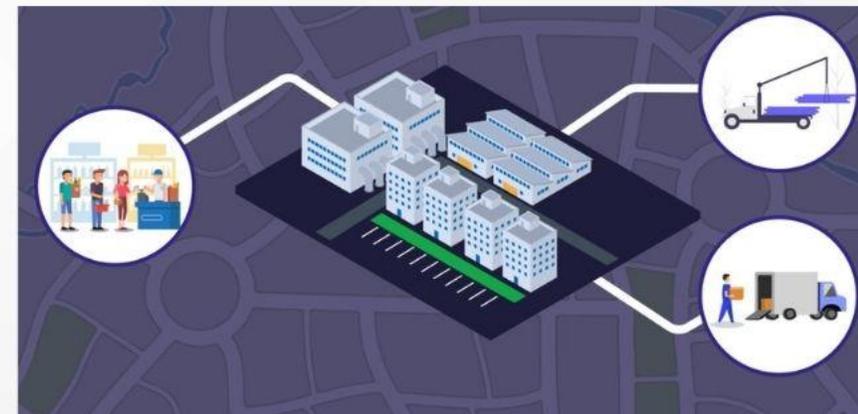
- Unlike simpler assignment problems, **GAP** allows for assigning multiple tasks to a single resource if the resource's capacity is not surpassed. This flexibility mirrors the complexity of real-world scenarios such as production planning and project scheduling, where resources may need to handle multiple tasks simultaneously.
- The **Generalized Assignment Problem** is widely applicable across fields like **manufacturing, telecommunications, transportation, and project management**. Its adaptability makes it an invaluable tool for addressing resource allocation challenges involving multiple tasks and resources with different capacities and costs.

PRACTICAL APPLICATIONS OF GAP

Manufacturing and Production Planning: Allocating machines to different production tasks to minimize production costs. Assigning workers to specific manufacturing operations considering their skills and capacity.

Project Scheduling: Assigning project tasks to available team members while minimizing costs and considering skill requirements.

Facility Location and Assignment: In logistics and supply chain management, GAP can be applied to decide the optimal location for facilities (such as warehouses or distribution centers) and assign tasks related to order fulfillment to these facilities.



MATHEMATICAL MODEL

Description	General Assignment Problem
Data / Parameters	<ul style="list-style-type: none">• n: Number of jobs to be assigned.• m: Number of agents.• Define ($n \geq m$) and $N = \{1, 2, \dots, n\}$
Decision	<ul style="list-style-type: none">• Designate multiple tasks to agents.
Objective	The goal is to minimize the total cost of assigning resources to tasks while satisfying the capacity constraints of each resource.
Constrains	<ul style="list-style-type: none">• Each job is assigned exactly to one agent.• The total resource requirement of the jobs assigned to an agent does not exceed the capacity of the agent.

MATHEMATICAL MODEL

The GAP may be formulated as a 0–1 integer linear programming (ILP) model. Let n be the number of tasks to be assigned to m agents. We define the requisite data elements as follows:

- c_{ij} = cost of task j being assigned to agent i .
- a_i = capacity of agent i .
- b_j = the requirement of task j to be performed.

Decision Variables

- $x_{ij} = \begin{cases} 1 & \text{if task } j \text{ is assigned to agent } i \\ 0 & \text{if not.} \end{cases}$

MATHEMATICAL MODEL

The 0-1 ILP model may then be written as:

Minimize:

$$\sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \quad (1)$$

The objective function (1) sums the costs of the assignments

Subject to:

$$\sum_{j=1}^n b_j x_{ij} \leq a_i, \quad \forall i \quad (2)$$

Constraint (2) enforces the resource limitation for each agent.

$$\sum_{i=1}^m x_{ij} = 1, \quad \forall j \in N \quad (3)$$

Constraint (3) ensures that each job is assigned to exactly one agent. We allow all data elements to be real (certain efficiencies follow if the data elements are assumed to be integral)

$$x_{ij} = 0 \text{ or } 1, \quad \forall i, j \quad (4)$$



PROBLEM EXAMPLE

Consider a small instance of the GAP with:

Agent Capacities: $a_1=7$, $a_2=5$

Task Requirement: $b_1=2$, $b_2=3$, $b_3=2$

Cost	Agent 1	Agent 2
Task 1	1	3
Task 2	2	1
Task 3	2	4



FEASIBLE SOLUTION

To have a feasible solution we need to meet the constraints which are:

Constraints:

- Each task must be assigned to exactly one agent. ✓
- The capacity of each agent must not be exceeded. ✓

Feasible Solution:

$x_{11}=1, x_{12}=0, x_{13}=1$ (Tasks T1 and T3 is assigned to agent 1)

$x_{21}=0, x_{22}=1, x_{23}=0$ (Task T1 is assigned to agent 2)

agent 1 capacity: $7-2-2 = 3$ ✓

agent 2 capacity = $5-3 = 2$ ✓

Objective Function Evaluation:

Total Profit = $4(T1) + 5(T3) + 6(T2) = 15$

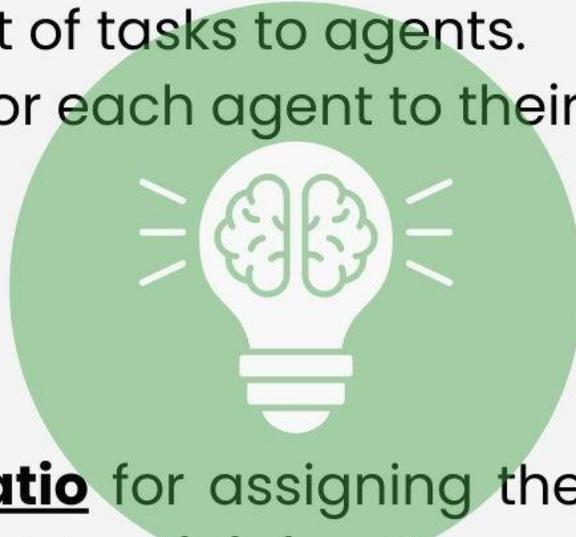
Total Cost = $1(T2) + 1(T1) + 2(T3) = 4$

DESCRIPTION OF HEURISTICS

HEURISTIC 1: GREEDY COST-CAPACITY RATIO HEURISTIC

Step 1: Initialization

- Start with an empty assignment of tasks to agents.
- Initialize remaining capacities for each agent to their respective maximum capacities.
- Initialize the total cost to 0.



Step 2: Task Selection

For each task:

- Compute the **cost-capacity ratio** for assigning the task to each available agent. The cost-capacity ratio is calculated by **dividing the cost of assigning the task to an agent by the remaining capacity of that agent**.
- Select the agent with the minimum cost-capacity ratio for the task.

DESCRIPTION OF HEURISTICS

HEURISTIC 1: GREEDY COST-CAPACITY RATIO HEURISTIC

Step 3: Task Assignment:

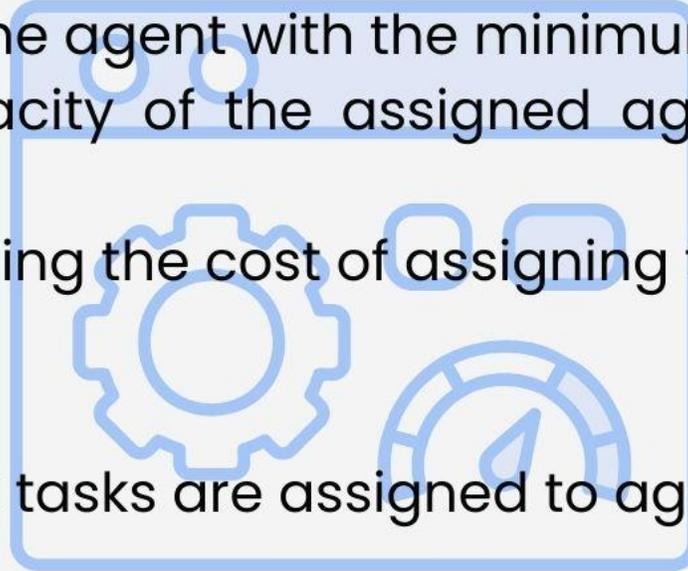
- Assign the selected task to the agent with the minimum cost-capacity ratio.
- Update the remaining capacity of the assigned agent by decrementing it by the requirement.
- Update the total cost by adding the cost of assigning the task to the selected agent.

Step 4: Repeat:

- Repeat steps 2 and 3 until all tasks are assigned to agents.

Step 5: Termination:

- Once all tasks are assigned, the algorithm terminates.



DESCRIPTION OF HEURISTICS

HEURISTIC 1: GREEDY COST-CAPACITY RATIO HEURISTIC

Example:

Let's consider a simple example with 3 tasks and 2 agents. Here are the details:

- **Tasks:** 3
- **Agents:** 2
- **Capacities:** Agent 1: 6, Agent 2: 5
- **Requirement:** Task 1: 3 Task 2: 2 Task 3: 3
- **Costs Matrix:**

	Agent 1	Agent 2
Task 1	10	5
Task 2	8	6
Tas 3	7	9

DESCRIPTION OF HEURISTICS

HEURISTIC 1: GREEDY COST-CAPACITY RATIO HEURISTIC

Tasks 1:

- **Cost-Capacity Ratio:** Agent 1 = $10/6 = 1.66$, Agent 2 = $5/6 = 0.866$
- Assign Task 1 to Agent 2 (min ratio 0.866).
- **Updated Capacities:** Agent 1: 6, Agent 2: 2
- **Total Cost:** 5

Tasks 2:

- **Cost-Capacity Ratio:** Agent 1 = $8/6 = 1.33$, Agent 2 = $6/2 = 3$
- Assign Task 2 to Agent 1 (min ratio 1.33).
- **Updated Capacities:** Agent 1: 3, Agent 2: 2
- **Total Cost:** $5 + 8 = 13$

Tasks 3:

- **Cost-Capacity Ratio:** Agent 1 = $7/3 = 2.33$, Agent 2 = $9/2 = 4.5$
- Assign Task 3 to Agent 1 (min ratio 2.33).
- **Updated Capacities:** Agent 1: 0, Agent 2: 2
- **Total Cost:** $13 + 7 = 20$

Final Assignment:

- Task 1 -> Agent 2
- Task 2 -> Agent 1
- Task 3 -> Agent 1

Total Cost: 20

DESCRIPTION OF HEURISTICS

LOCAL SEARCH : TASK REASSIGNATION

Step 1: Initial Solution Generation

- First, generate an initial solution using the Greedy Cost-Capacity Ratio heuristic. This solution will have each task assigned to an agent based on the minimum cost-capacity ratio.

Step 2: Neighbor Solution Generation:

- Identify "neighbor" solutions by **reassigning one task to a different agent.**

`move1(l, i; k)` =reassign task $i \in X$ of agent l to agent k

$$\Delta z = c_{l,i} - c_{k,i}$$

$$\Delta z > 0 \quad \text{NO}$$

$$\Delta z < 0 \quad \text{YES}$$

Step 3: Evaluate Neighbor Solutions:

- Calculate the total cost of each neighbor solution. Compare these costs to the total cost of the current solution.

DESCRIPTION OF HEURISTICS

LOCAL SEARCH : TASK REASSIGNATION

Step 4: Acceptance Criterion:

- If a neighbor solution has a lower total cost than the current solution, accept the neighbor solution as the new current solution. If no better neighbor is found, the current solution remains unchanged.

Step 5 : Iteration:

- Repeat the process of generating neighbor solutions and evaluating them until no further improvements can be made, or a predefined number of iterations is reached.

Step 6: Termination:

- The algorithm terminates when no better neighbor solution is found, indicating a locally optimal solution has been achieved.

DESCRIPTION OF HEURISTICS

LOCAL SEARCH : TASK REASSIGNATION

Local Search Process:

1. Generate Neighbor Solutions:

- $\text{move}_1(2, 1; 1) =$ Not possible (capacity exceeded, **agent 1** has 1 of capacity left and the requirement of this task is **3**).
- $\text{move}_1(1, 2; 2) = -8+6 = -2$ **BEST**
- $\text{move}_1(1, 3; 2) =$ Not possible (capacity exceeded, **agent 2** has 2 of capacity left and the requirement of this task is **3**).

2. Evaluate Neighbor Solutions:

- Only valid neighbor solution with improved cost: Task 2 to Agent 2



DESCRIPTION OF HEURISTICS

LOCAL SEARCH : TASK REASSIGNATION

3. Update Solution:

Accept the neighbor solution with Task 2 reassigned to Agent 2:

Updated Capacities: Agent 1: 3, Agent 2: 0

Total Cost: 5 (Task 1) + 6 (Task 2) + 7 (Task 3) = 18

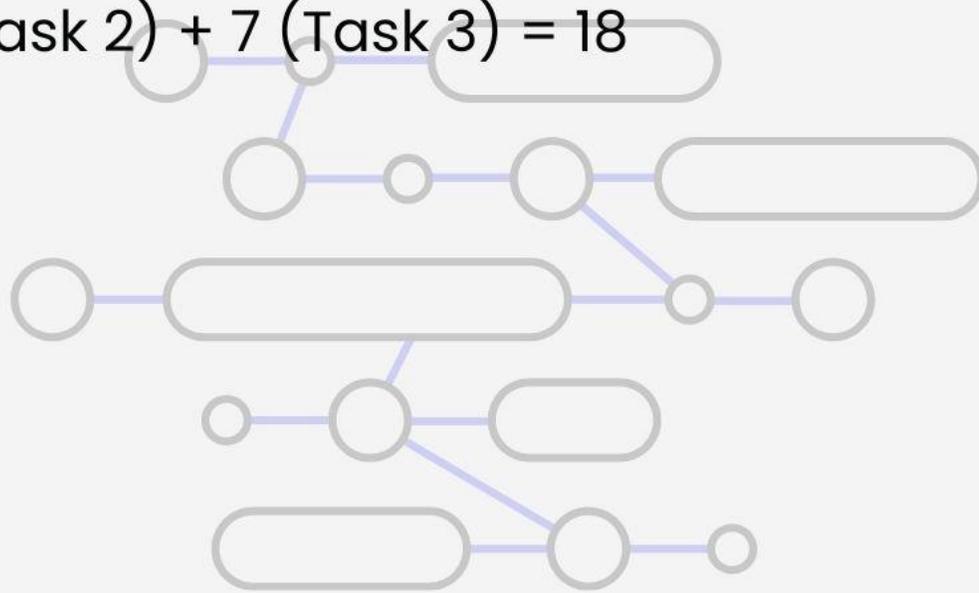
New Assignment:

Task 1 -> Agent 2

Task 2 -> Agent 2

Task 3 -> Agent 1

Total Cost: 18



4. Iteration:

Continue searching for neighbor solutions. For simplicity, assume no further improvement is found.

DESCRIPTION OF HEURISTICS

LOCAL SEARCH : TASK REASSIGNATION

5. Termination:

The local search terminates, and the final assignment is:

Task 1 -> Agent 2

Task 2 -> Agent 2

Task 3 -> Agent 1

Total Cost: 18



DESCRIPTION OF HEURISTICS

HEURISTIC 2: GREEDY CAPACITY BALANCING ASSIGNMENT

Step 1: Initialization

- Start with all tasks unassigned.
- Initialize the remaining capacity of each agent based on their initial capacity.

Step 2: Iterative Task Assignment

- For each task, determine the best agent to assign it to. The "best" agent is chosen based on a **metric** ($\text{cost}/(\text{remaining capacity} + 1)$) that combines the cost of assignment and the remaining capacity of the agent.
- Assign the task to the agent that has the lowest cost per remaining capacity (cost divided by remaining capacity plus one).
- Update the remaining capacity of the chosen agent.

DESCRIPTION OF HEURISTICS

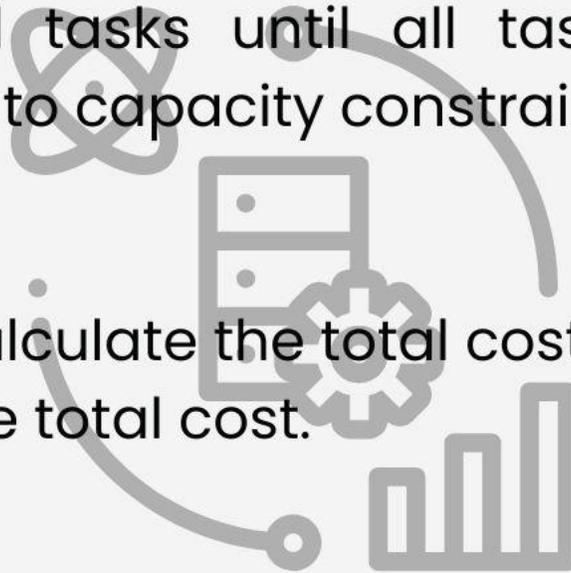
HEURISTIC 2: GREEDY CAPACITY BALANCING ASSIGNMENT

Step 3: Repeat

- Continue the process for all tasks until all tasks are assigned or no feasible assignment can be made due to capacity constraints.

Step 4: Completion

- Once all tasks are assigned, calculate the total cost of the assignment.
- Output the assignment and the total cost.



DESCRIPTION OF HEURISTICS

HEURISTIC 2: GREEDY CAPACITY BALANCING ASSIGNMENT

Example:

Let's consider a simple example with 3 tasks and 2 agents. Here are the details:

- **Tasks:** 4
- **Agents:** 3
- **Capacities:** Agent 1: 6, Agent 2: 5, Agent 3:6
- **Requirement:** Task 1: 3 Task 2: 2 Task 3: 3 Task 4:2
- **Costs Matrix:**

	Agent 1	Agent 2	Agent 3
Task 1	10	5	7
Task 2	8	6	9
Task 3	7	9	4
Task 4	6	3	8

DESCRIPTION OF HEURISTICS

HEURISTIC 2: GREEDY CAPACITY BALANCING ASSIGNMENT

Tasks 1:

Calculate metric (cost / (remaining capacity + 1)) for Task 1:

- **Agent 1:** $10 / (6 + 1) = 1.42$
- **Agent 2:** $5 / (5 + 1) = 0.833$
- **Agent 3:** $7 / (6 + 1) = 1$
- Assign Task 1 to Agent 2 (lowest metric: 0.833)
- **Update remaining capacities:** Agent 1: 6, Agent 2: 2, Agent 3: 6
- **Assigned tasks:** Task 1 → Agent 2

Tasks 2:

- **Agent 1:** $8 / (6 + 1) = 1.14$
- **Agent 2:** $6 / (2 + 1) = 3$
- **Agent 3:** $9 / (6 + 1) = 1.28$
- Assign Task 2 to Agent 1 (lowest metric: 1.14)
- **Update remaining capacities:** Agent 1: 4, Agent 2: 2, Agent 3: 6
- **Assigned tasks:** Task 1 → Agent 2, Task 2 → Agent 1

DESCRIPTION OF HEURISTICS

HEURISTIC 2: GREEDY CAPACITY BALANCING ASSIGNMENT

Tasks 3:

- **Agent 1:** $7 / (4 + 1) = 1.4$
- **Agent 2:** $9 / (2 + 1) = 3$
- **Agent 3:** $4 / (6 + 1) = 0.57$
- Assign Task 3 to Agent 3 (lowest metric: 0.57)
- **Update remaining capacities:** Agent 1: 4, Agent 2: 2, Agent 3: 3
- **Assigned tasks:** Task 1 -> Agent 2, Task 2 -> Agent 1, Task 3 -> Agent 3

Tasks 4:

- **Agent 1:** $6 / (4 + 1) = 1.2$
- **Agent 2:** $3 / (2 + 1) = 1$
- **Agent 3:** $8 / (3 + 1) = 2$
- Assign Task 4 to Agent 2 (lowest metric: 1)
- **Update remaining capacities:** Agent 1: 4, Agent 2: 0, Agent 3: 3
- **Assigned tasks:** Task 1 -> Agent 2, Task 2 -> Agent 1, Task 3 -> Agent 3, Task 4 -> Agent 2

Final Assignment

- Task 1 -> Agent 2
- Task 2 -> Agent 1
- Task 3 -> Agent 3
- Task 4 -> Agent 2

Total Cost: 20

EXPERIMENTATION

Experiments for the GAP

For the experiments we will approach 3 different sizes of data:

Data

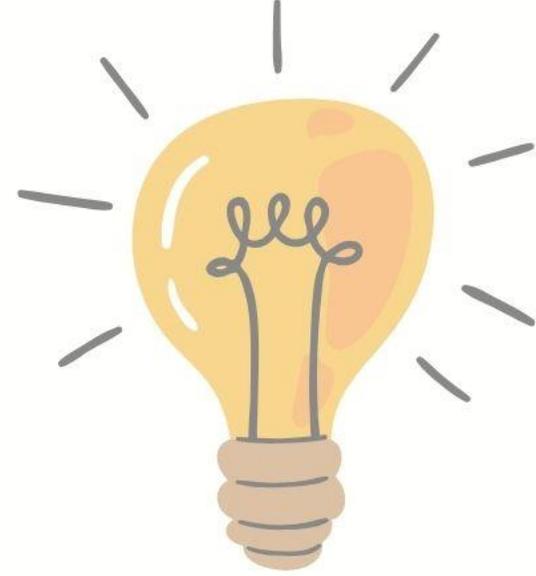
Small: Set of size $n = 1000$ – 50 tasks, 20 agents

Medium: Set of size $n = 10,000$ – 200 tasks, 50 agents

Large: Set of size $n = 100,000$ – 500 tasks, 200 agents

- For every task the cost will be generated randomly between 0 and 101
- For every agent the capacity will be generated randomly

There will be generated 20 instances for every size of data (**small**, **medium**, and **large**). Then in every instance of every size, we will be applying the two heuristics (**CH1**: Greedy Cost-Capacity Ratio and **CH2**: Greedy Capacity Balancing Assignment) as well the local search algorithms in both of them, to visualize and compare the minimum cost on every instance and compare the results so we can select which was a better a more efficient Heuristic as well the local search algorithm.



EXPERIMENT 1: CH1 VS CH2

Small		CH1 vs CH2 (with NO local search)					
Instance	CH1_OF	CH1_time (cpu sec)	CH2_OF	CH2_time (cpu sec)	ABS CH1-CH2	REL IMP OF CH1 over CH2	REL IMP OF CH1 over CH2
Small_01	367	0.008	348	0.008	19	-0.054597701	0.051771117
Small_02	305	0.008	305	0.008	0	0	0
Small_03	369	0.009	367	0.009	2	-0.005449591	0.005420054
Small_04	322	0.008	309	0.008	13	-0.042071197	0.040372671
Small_05	321	0.009	284	0.008	37	-0.13028169	0.115264798
Small_06	374	0.009	352	0.009	22	-0.0625	0.058823529
Small_07	394	0.008	394	0.009	0	0	0
Small_08	347	0.009	346	0.008	1	-0.002890173	0.002881844
Small_09	352	0.008	347	0.008	5	-0.014409222	0.014204545
Small_10	356	0.009	338	0.009	18	-0.053254438	0.050561798
Small_11	421	0.018	399	0.009	22	-0.055137845	0.052256532
Small_12	396	0.009	370	0.009	26	-0.07027027	0.065656566
Small_13	314	0.009	303	0.008	11	-0.03630363	0.035031847
Small_14	316	0.011	311	0.009	5	-0.01607717	0.015822785
Small_15	340	0.009	326	0.008	14	-0.042944785	0.041176471
Small_16	293	0.008	295	0.008	-2	0.006779661	-0.006825939
Small_17	294	0.01	292	0.008	2	-0.006849315	0.006802721
Small_18	345	0.009	343	0.008	2	-0.005830904	0.005797101
Small_19	304	0.009	289	0.009	15	-0.051903114	0.049342105
Small_20	350	0.009	338	0.008	12	-0.035502959	0.034285714
Average Time		0.0093		0.0084			

Medium		CH1 vs CH2 (with NO local search)					
Instance	CH1_OF	CH1_time (cpu sec)	CH2_OF	CH2_time (cpu sec)	ABS CH1-CH2	REL IMP OF CH1 over CH2	REL IMP OF CH2 over CH1
Medium_01	716	0.321	706	0.307	10	-0.014164306	0.01396648
Medium_02	642	0.317	642	0.313	0	0	0
Medium_03	588	0.319	583	0.317	5	-0.008576329	0.008503401
Medium_04	605	0.336	579	0.32	26	-0.044905009	0.042975207
Medium_05	634	0.316	632	0.319	2	-0.003164557	0.003154574
Medium_06	677	0.313	669	0.313	8	-0.011958146	0.011816839
Medium_07	661	0.307	655	0.32	6	-0.009160305	0.009077156
Medium_08	662	0.311	657	0.313	5	-0.00761035	0.00755287
Medium_09	722	0.31	722	0.316	0	0	0
Medium_10	683	0.316	681	0.319	2	-0.002936858	0.002928258
Medium_11	762	0.316	730	0.316	32	-0.043835616	0.041994751
Medium_12	736	0.309	732	0.317	4	-0.005464481	0.005434783
Medium_13	691	0.317	686	0.319	5	-0.00728863	0.00723589
Medium_14	583	0.309	581	0.316	2	-0.003442341	0.003430532
Medium_15	762	0.306	760	0.317	2	-0.002631579	0.002624672
Medium_16	703	0.319	703	0.319	0	0	0
Medium_17	651	0.313	651	0.316	0	0	0
Medium_18	662	0.305	654	0.317	8	-0.012232416	0.012084592
Medium_19	678	0.32	673	0.307	5	-0.007429421	0.007374631
Medium_20	776	0.306	767	0.317	9	-0.011734029	0.011597938
Average Time		0.3143		0.3159			

Large		CH1 vs CH2 (with NO local search)					
Instance	CH1_OF	CH1_time (cpu sec)	CH2_OF	CH2_time (cpu sec)	ABS CH1-CH2	REL IMP OF CH1 over CH2	REL IMP OF CH2 over CH1
Large_01	694	0.317	691	0.336	3	-0.004341534	0.004322767
Large_02	703	0.336	703	0.309	0	0	0
Large_03	686	0.313	686	0.307	0	0	0
Large_04	683	0.307	682	0.313	1	-0.001466276	0.001464129
Large_05	703	0.317	702	0.307	1	-0.001424501	0.001422475
Large_06	679	0.336	672	0.309	7	-0.010416667	0.010309278
Large_07	688	0.309	695	0.316	-7	0.010071942	-0.010174419
Large_08	692	0.307	692	0.313	0	0	0
Large_09	686	0.316	686	0.313	0	0	0
Large_10	711	0.31	711	0.316	0	0	0
Large_11	673	0.336	709	0.307	-36	0.05077574	-0.053491828
Large_12	713	0.317	673	0.309	40	-0.059435364	0.056100982
Large_13	684	0.306	682	0.336	2	-0.002932551	0.002923977
Large_14	702	0.336	702	0.313	0	0	0
Large_15	742	0.316	742	0.316	0	0	0
Large_16	727	0.317	726	0.309	1	-0.00137741	0.001375516
Large_17	695	0.309	695	0.313	0	0	0
Large_18	677	0.306	677	0.309	0	0	0
Large_19	683	0.336	683	0.336	0	0	0
Large_20	701	0.313	699	0.313	2	-0.00286123	0.002853067
Average Time		0.318		0.315			

EXPERIMENT 2: CH1 VS CH1_LS

Small		CH1 vs CH1_LS		(with local search)			
Instance	CH1_OF	CH1_time (cpu sec)	CH1_LS	CH1LS_time (cpu sec)	ABS CH1-CH1LS	REL IMP OF CH1 over CH1LS	REL IMP OF CH1s over CH1
Small_01	367	0.008	285	2.77	82	-0.287719298	0.223433243
Small_02	305	0.008	259	2.77	46	-0.177606178	0.150819672
Small_03	369	0.009	296	2.789	73	-0.246621622	0.197831978
Small_04	322	0.008	240	2.742	82	-0.341666667	0.254658385
Small_05	321	0.009	245	2.778	76	-0.310204082	0.236760125
Small_06	374	0.009	260	2.769	114	-0.438461538	0.304812834
Small_07	394	0.008	347	2.803	47	-0.135446686	0.11928934
Small_08	347	0.009	282	2.734	65	-0.230496454	0.187319885
Small_09	352	0.008	290	2.815	62	-0.213793103	0.176136364
Small_10	356	0.009	297	2.758	59	-0.198653199	0.165730337
Small_11	421	0.018	288	2.798	133	-0.461805556	0.315914489
Small_12	396	0.009	295	2.849	101	-0.342372881	0.255050505
Small_13	314	0.009	286	2.748	28	-0.097902098	0.089171975
Small_14	316	0.011	280	2.771	Cerrar	-0.128571429	0.113924051
Small_15	340	0.009	268	2.733	72	-0.268656716	0.211764706
Small_16	293	0.008	240	2.734	53	-0.220833333	0.180887372
Small_17	294	0.01	233	2.566	61	-0.261802575	0.207482993
Small_18	345	0.009	278	2.787	67	-0.241007194	0.194202899
Small_19	304	0.009	246	3.044	58	-0.235772358	0.190789474
Small_20	350	0.009	266	2.781	84	-0.315789474	0.24
Average Time		0.0093		2.77695			

Medium		CH1 vs CH1_LS		(with local search)			
Instance	CH1_OF	CH1_time (cpu sec)	CH1_LS	CH1LS_time (cpu sec)	ABS CH1-CH1LS	REL IMP OF CH1 over CH1LS	REL IMP OF CH1s over CH1
Medium_01	716	0.321	540	117.32	176	-0.325925926	0.245810056
Medium_02	642	0.317	489	110.712	153	-0.312883436	0.238317757
Medium_03	588	0.319	469	108.927	119	-0.253731343	0.202380952
Medium_04	605	0.336	322	110.724	283	-0.878881988	0.467768595
Medium_05	634	0.316	516	108.445	118	-0.228682171	0.186119874
Medium_06	677	0.313	515	109.353	162	-0.314563107	0.23929099
Medium_07	661	0.307	471	108.788	190	-0.403397028	0.287443268
Medium_08	662	0.311	488	109.184	174	-0.356557377	0.262839879
Medium_09	722	0.31	532	108.345	190	-0.357142857	0.263157895
Medium_10	683	0.316	534	109.647	149	-0.279026217	0.218155198
Medium_11	762	0.316	483	108.767	279	-0.577639752	0.366141732
Medium_12	736	0.309	537	108.864	199	-0.370577281	0.270380435
Medium_13	691	0.317	509	109.301	182	-0.357563851	0.263386397
Medium_14	583	0.309	467	109.008	116	-0.248394004	0.19897084
Medium_15	762	0.306	555	109.516	207	-0.372972973	0.271653543
Medium_16	703	0.319	520	109.382	183	-0.351923077	0.260312945
Medium_17	651	0.313	498	109.583	153	-0.307228916	0.235023041
Medium_18	662	0.305	504	109.569	158	-0.313492063	0.238670695
Medium_19	678	0.32	523	109.939	155	-0.296367113	0.228613569
Medium_20	776	0.306	522	109.916	254	-0.486590038	0.327319588
Average Time		0.3143		109.7645			

Large		CH1 vs CH1_LS		(with local search)			
Instance	CH1_OF	CH1_time (cpu sec)	CH1_LS	CH1LS_time (cpu sec)	ABS CH1-CH1LS	REL IMP OF CH1 over CH1LS	REL IMP OF CH1s over CH1
Large_01	694	0.317	567	449.38	127	-0.223985891	0.182997118
Large_02	703	0.336	498	481.58	205	-0.411646586	0.291607397
Large_03	686	0.313	567	434.32	119	-0.209876543	0.173469388
Large_04	683	0.307	534	465.62	149	-0.279026217	0.218155198
Large_05	703	0.317	467	426.54	236	-0.505353319	0.335704125
Large_06	679	0.336	455	481.58	224	-0.492307692	0.329896907
Large_07	688	0.309	598	516.7	90	-0.150501672	0.130813953
Large_08	692	0.307	573	449.38	119	-0.207678883	0.171965318
Large_09	686	0.316	520	427.64	166	-0.319230769	0.241982507
Large_10	711	0.31	522	550.54	189	-0.362068966	0.265822785
Large_11	673	0.336	455	465.62	218	-0.479120879	0.323922734
Large_12	713	0.317	532	427.64	181	-0.340225564	0.253856942
Large_13	684	0.306	566	519.46	118	-0.208480565	0.17251462
Large_14	702	0.336	571	481.58	131	-0.229422067	0.186609687
Large_15	742	0.316	491	556.85	251	-0.511201629	0.338274933
Large_16	727	0.317	543	434.32	184	-0.388858195	0.253094911
Large_17	695	0.309	512	465.62	183	-0.357421875	0.263309353
Large_18	677	0.306	444	519.46	233	-0.524774775	0.344165436
Large_19	683	0.336	505	489.34	178	-0.352475248	0.260614934
Large_20	701	0.313	555	436.61	146	-0.263063063	0.208273894
Average Time		0.318		473.989			

EXPERIMENT 3: CH2 VS CH2_LS

Small		CH2 vs CH2_LS		(with local search)			
Instance	CH2_OF	CH2_time (cpu sec)	CH2_LS	CH2LS_time (cpu sec)	ABS CH2-CH2LS	REL IMP OF CH2 over CH2LS	REL IMP OF CH2ls over CH2
Small_01	348	0.008	258	0.05	90	-0.348837209	0.25862069
Small_02	305	0.008	247	0.04	58	-0.234817814	0.190163934
Small_03	367	0.009	276	0.04	91	-0.329710145	0.247956403
Small_04	309	0.008	240	0.04	69	-0.2875	0.223300971
Small_05	284	0.008	245	0.82	39	-0.159183673	0.137323944
Small_06	352	0.009	226	0.08	126	-0.557522124	0.357954545
Small_07	394	0.009	329	0.06	65	-0.197568389	0.164974619
Small_08	346	0.008	281	0.05	65	-0.231316726	0.187861272
Small_09	347	0.008	290	0.05	57	-0.196551724	0.16426513
Small_10	338	0.009	296	0.05	42	-0.141891892	0.124260355
Small_11	399	0.009	288	0.05	111	-0.385416667	0.278195489
Small_12	370	0.009	295	0.05	75	-0.254237288	0.202702703
Small_13	303	0.008	282	0.02	21	-0.074468085	0.069306931
Small_14	311	0.009	277	0.04	34	-0.122743682	0.109324759
Small_15	326	0.008	246	0.04	80	-0.325203252	0.245398773
Small_16	295	0.008	240	0.03	55	-0.229166667	0.186440678
Small_17	292	0.008	233	0.04	59	-0.253218884	0.202054795
Small_18	343	0.008	278	0.04	65	-0.23381295	0.189504373
Small_19	289	0.009	193	0.05	96	-0.497409326	0.332179931
Small_20	338	0.008	265	0.04	73	-0.275471698	0.215976331
Average Time		0.0084		0.084			

Medium		CH2 vs CH2_LS		(with local search)			
Instance	CH2_OF	CH2_time (cpu sec)	CH2_LS	CH2LS_time (cpu sec)	ABS CH2-CH2LS	REL IMP OF CH2 over CH2LS	REL IMP OF CH2ls over CH2
Medium_01	706	0.307	538	5.87	168	-0.312267658	0.23796034
Medium_02	642	0.313	489	5.2	153	-0.312883436	0.238317757
Medium_03	583	0.317	469	5.53	114	-0.243070362	0.195540309
Medium_04	579	0.32	480	4.51	99	-0.20625	0.170984456
Medium_05	632	0.319	515	5.55	117	-0.227184466	0.185126582
Medium_06	669	0.313	515	6.7	154	-0.299029126	0.23019432
Medium_07	655	0.32	471	7.06	184	-0.390658174	0.280916031
Medium_08	657	0.313	478	6.36	179	-0.374476987	0.272450533
Medium_09	722	0.316	495	6.34	227	-0.458585859	0.314404432
Medium_10	681	0.319	528	6.64	153	-0.289772727	0.224669604
Medium_11	730	0.316	473	7.07	257	-0.543340381	0.352054795
Medium_12	732	0.317	533	6.93	199	-0.373358349	0.271857923
Medium_13	686	0.319	509	6.32	177	-0.347740668	0.258017493
Medium_14	581	0.316	467	5.41	114	-0.244111349	0.196213425
Medium_15	760	0.317	549	8.13	211	-0.384335155	0.277631579
Medium_16	703	0.319	519	6.41	184	-0.354527938	0.26173542
Medium_17	651	0.316	494	8.39	157	-0.317813765	0.241167435
Medium_18	654	0.317	500	6.54	154	-0.308	0.235474006
Medium_19	673	0.307	521	6.54	152	-0.291746641	0.225854383
Medium_20	767	0.317	518	7.99	249	-0.480694981	0.32464146
Average Time		0.3159		6.4745			

Large		CH2 vs CH2_LS		(with local search)			
Instance	CH2_OF	CH2_time (cpu sec)	CH2_LS	CH2LS_time (cpu sec)	ABS CH2-CH2LS	REL IMP OF CH2 over CH2LS	REL IMP OF CH2ls over CH2
Large_01	691	0.336	583	253.37	108	-0.185248714	0.156295224
Large_02	703	0.309	586	281.58	117	-0.199658703	0.166429587
Large_03	686	0.307	578	265.62	108	-0.186851211	0.157434402
Large_04	682	0.313	580	249.38	102	-0.175862069	0.149560117
Large_05	702	0.307	596	269.54	106	-0.177852349	0.150997151
Large_06	672	0.309	569	259.16	103	-0.181019332	0.15327381
Large_07	695	0.316	596	227.64	99	-0.166107383	0.142446043
Large_08	692	0.313	583	261.58	109	-0.186963979	0.157514451
Large_09	686	0.313	585	226.54	101	-0.172649573	0.147230321
Large_10	711	0.316	582	340.54	129	-0.221649485	0.181434599
Large_11	709	0.307	570	233.84	139	-0.243859649	0.196050776
Large_12	673	0.309	574	316.7	99	-0.172473868	0.147102526
Large_13	682	0.336	577	264.78	105	-0.181975737	0.153958944
Large_14	702	0.313	570	293.31	132	-0.231578947	0.188034188
Large_15	742	0.316	586	356.85	156	-0.266211604	0.210242588
Large_16	726	0.309	579	319.46	147	-0.25388601	0.202479339
Large_17	695	0.313	569	279.91	126	-0.221441125	0.181294964
Large_18	677	0.309	579	236.61	98	-0.16925734	0.144756278
Large_19	683	0.336	576	234.32	107	-0.185763889	0.156661786
Large_20	699	0.313	588	289.34	111	-0.18877551	0.158798283
Average Time		0.315		273.0035			

EXPERIMENT 4: CH1_LS VS CH2_LS

Small		CH1_LS vs CH2_LS		(with local search)					
Instance	CH1_LS	CH1LS_time (cpu sec)	CH2_LS	CH2LS_time (cpu sec)	ABS CH1LS-CH2LS	REL IMP OF CH2LS over CH2LS	REL IMP OF CH2LS over CH1LS		
Small_01	285	2.77	258	0.05	27	-0.104651163	0.094736842		
Small_02	259	2.77	247	0.04	12	-0.048582996	0.046332046		
Small_03	296	2.789	276	0.04	20	-0.072463768	0.067567568		
Small_04	240	2.742	240	0.04	0	0	0		
Small_05	245	2.778	245	0.82	0	0	0		
Small_06	260	2.769	226	0.08	34	-0.150442478	0.130769231		
Small_07	347	2.803	329	0.06	18	-0.054711246	0.051873199		
Small_08	282	2.734	281	0.05	1	-0.003558719	0.003546099		
Small_09	290	2.815	290	0.05	0	0	0		
Small_10	297	2.758	296	0.05	1	-0.003378378	0.003367003		
Small_11	288	2.798	288	0.05	0	0	0		
Small_12	295	2.849	295	0.05	0	0	0		
Small_13	286	2.748	282	0.02	4	-0.014184397	0.013986014		
Small_14	280	2.771	277	0.04	3	-0.010830325	0.010714286		
Small_15	268	2.733	246	0.04	22	-0.089430894	0.082089552		
Small_16	240	2.734	240	0.03	0	0	0		
Small_17	233	2.566	233	0.04	0	0	0		
Small_18	278	2.787	278	0.04	0	0	0		
Small_19	246	3.044	193	0.05	53	-0.274611399	0.215447154		
Small_20	266	2.781	265	0.04	1	-0.003773585	0.003759398		
Average Time		2.77695		0.084					

Medium		CH1_LS vs CH2_LS		(with local search)					
Instance	CH1_LS	CH1LS_time (cpu sec)	CH2_LS	CH2LS_time (cpu sec)	BS CH1LS-CH2LS	REL IMP OF CH2LS over CH2LS	REL IMP OF CH2LS over CH1LS		
Medium_01	540	117.32	538	5.87	2	-0.003717472	0.003703704		
Medium_02	489	110.712	489	5.2	0	0	0		
Medium_03	469	108.927	469	5.53	0	0	0		
Medium_04	322	110.724	480	4.51	-158	0.329166667	-0.49068323		
Medium_05	516	108.445	515	5.55	1	-0.001941748	0.001937984		
Medium_06	515	109.353	515	6.7	0	0	0		
Medium_07	471	108.788	471	7.06	0	0	0		
Medium_08	488	109.184	478	6.36	10	-0.020920502	0.020491803		
Medium_09	532	108.345	495	6.34	37	-0.074747475	0.069548872		
Medium_10	534	109.647	528	6.64	6	-0.011363636	0.011235955		
Medium_11	483	108.767	473	7.07	10	-0.021141649	0.020703934		
Medium_12	537	108.864	533	6.93	4	-0.00750469	0.00744879		
Medium_13	509	109.301	509	6.32	0	0	0		
Medium_14	467	109.008	467	5.41	0	0	0		
Medium_15	555	109.516	549	8.13	6	-0.010928962	0.010810811		
Medium_16	520	109.382	519	6.41	1	-0.001926782	0.001923077		
Medium_17	498	109.583	494	8.39	4	-0.008097166	0.008032129		
Medium_18	504	109.569	500	6.54	4	-0.008	0.007936508		
Medium_19	523	109.939	521	6.54	2	-0.003838772	0.003824092		
Medium_20	522	109.916	518	7.99	4	-0.007722008	0.007662835		
Average Time		109.7645		6.4745					

Large		CH1_LS vs CH2_LS		(with local search)					
Instance	CH1_LS	CH1LS_time (cpu sec)	CH2_LS	CH2LS_time (cpu sec)	BS CH1LS-CH2LS	REL IMP OF CH2LS over CH2LS	REL IMP OF CH2LS over CH1LS		
Large_01	567	449.38	583	253.37	-16	0.027444254	-0.028218695		
Large_02	498	481.58	586	281.58	-88	0.150170648	-0.176706827		
Large_03	567	434.32	567	265.62	0	0	0		
Large_04	534	465.62	580	249.38	-46	0.079310345	-0.086142322		
Large_05	467	426.54	596	269.54	-129	0.216442953	-0.276231263		
Large_06	455	481.58	569	259.16	-114	0.200351494	-0.250549451		
Large_07	598	516.7	596	227.64	2	-0.003355705	0.003344482		
Large_08	573	449.38	583	261.58	-10	0.017152659	-0.017452007		
Large_09	520	427.64	585	226.54	-65	0.111111111	-0.125		
Large_10	522	550.54	582	340.54	-60	0.103092784	-0.114942529		
Large_11	455	465.62	570	233.84	-115	0.201754386	-0.252747253		
Large_12	532	427.64	532	316.7	0	0	0		
Large_13	566	519.46	577	264.78	-11	0.019064125	-0.019434629		
Large_14	571	481.58	570	293.31	1	-0.001754386	0.001751313		
Large_15	491	556.85	586	356.85	-95	0.162116041	-0.193482688		
Large_16	543	434.32	543	319.46	0	0	0		
Large_17	512	465.62	569	279.91	-57	0.100175747	-0.111328125		
Large_18	444	519.46	444	236.61	0	0	0		
Large_19	505	489.34	576	234.32	-71	0.123263889	-0.140594059		
Large_20	555	436.61	588	289.34	-33	0.056122449	-0.059459459		
Average Time		473.989		273.0035					

EXPERIMENT AND RESULTS

Experiment Number 1: CH1 vs CH2

In this case, for small instances, it can be observed that Heuristic Number 2 (Greedy Capacity Balancing Assignment) delivered better results in both costs and execution time compared to Heuristic Number 1 (Greedy Cost-Capacity Ratio). This suggests that for smaller problem sizes, Heuristic Number 2 is more efficient at balancing agent capacities and minimizing costs quickly.

However, in the case of **medium** and **large** instances, both heuristics yielded similar results. In several instances, the data were even identical, despite the heuristics having different approaches.

The performance of both heuristics highlights their utility in practical applications. They both work very well, showcasing the ability to handle varying problem sizes efficiently.

EXPERIMENT AND RESULTS

Experiment Number 2: CHI vs CHI_LS

We can observe that the local search algorithm achieved better results in terms of costs compared to *Heuristic number 1* (H1), but with increased execution time. The rise in execution time is primarily due to the numerous iterations the algorithm must perform to discover an improved solution.

This iterative process involves exploring the solution space more thoroughly, which naturally extends the computational effort required. It can be concluded that for every instance size tested, the local search algorithm consistently produced better results in terms of cost reduction than **Heuristic number 1**.

This indicates that the local search method is more effective at navigating the solution space to find optimal or near-optimal solutions, even though it requires more computational resources.

EXPERIMENT AND RESULTS

Experiment Number 3: CH2 vs CH2_LS

In this case, the results from the local search algorithm were superior to those from **Heuristic Number 2** (Greedy Capacity Balancing Assignment). The local search algorithm minimized costs across every instance size, demonstrating a clear advantage in cost efficiency.

However, similar to the findings in experiment **number 2**, this enhancement in cost optimization came with an increased execution time.

Experiment Number 4: CH1_LS vs CH2_LS

In fact, Heuristic 1's local search algorithm almost doubled the execution time for every instance size when compared to Heuristic 2. This substantial difference in execution time suggests that while both algorithms are effective in minimizing costs, their efficiency in terms of computational resources varies considerably.

