Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Operations Research 33 (2006) 1380–1398
computers & operations research
www.elsevier.com/locate/cor

Upper bounds and exact algorithms for p-dispersion problems

David Pisinger*

DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark

Abstract

The p-dispersion-sum problem is the problem of locating p facilities at some of n predefined locations, such that the distance sum between the p facilities is maximized. The problem has applications in telecommunication (where it is desirable to disperse the transceivers in order to minimize interference problems), and in location of shops and service-stations (where the mutual competition should be minimized).

A number of fast upper bounds are presented based on Lagrangian relaxation, semidefinite programming and reformulation techniques. A branch-and-bound algorithm is then derived, which at each branching node is able to compute the reformulation-based upper bound in $O(n)$ time. Computational experiments show that the algorithm may solve geometric problems of size up to $n = 90$, and weighted geometric problems of size $n = 250$.

The related p-dispersion problem is the problem of locating p facilities such that the minimum distance between two facilities is as large as possible. New formulations and fast upper bounds are presented, and it is discussed whether a similar framework as for the p-dispersion sum problem can be used to tighten the upper bounds. A solution algorithm based on transformation of the p-dispersion problem to the p-dispersion-sum problem is finally presented, and its performance is evaluated through several computational experiments.

Keywords: Locational analysis; Upper bounds; Branch-and-bound

1. Introduction

We consider the problem of establishing p facilities at some of n predefined locations. The distance between two facilities i and j is given by a square matrix (d_{ij}), $i, j = 1, \ldots, n$. In the *p-dispersion-sum problem* (PDSP) the objective is to maximize the distance sum between the p established facilities. Since

* Tel.: +45 35 32 13 54; fax: +45 35 32 14 01.
E-mail address: pisinger@diku.dk (D. Pisinger).

0305-0548/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2004.09.033

the number of selected facilities is constant, maximizing the distance sum is equivalent to maximizing the average distance between facilities. A different variant of the problem called the *p-dispersion problem* appears when the objective is to maximize the minimum distance between two established facilities.

Both variants of the problem have several applications in telecommunication and warehouse location. In telecommunication one may for example wish to disperse radio transceivers to service cellular phones in order to minimize interference problems. In the case of locating branches of a chain, one wishes to minimize mutual competition between similar shops or service-stations. Moreover, the problems have several applications in military defense, since it is common practice to scatter ones installations in order to make it more difficult for the enemy to disarm them. The PDSP is thus also known as the *p-defense-sum problem*. The problem may also find applications in data mining, where diverse method of data clustering are being explored [1]. In graph theory, the *heaviest subgraph* problem considers a weighted graph (V, E, d). The problem is to select a node subset $K \subseteq V$ of cardinality $|K| = p$ such that the weight of the subgraph induced by K is maximized. This problem is obviously equivalent to the PDSP.

Both of the problems are $\mathcal{NP}$-hard which easily can be proved by reduction from the clique problem [2,3]. Even if the distance matrix satisfies the triangle inequality, the problems remain $\mathcal{NP}$-hard [2,3]. Ravi et al. [4] showed that the p-dispersion problem cannot be approximated by a fixed ratio ρ unless $\mathcal{NP}=\mathcal{P}$. If the triangle inequality is satisfied, an approximation ratio of $\rho = 2$ can be obtained, and (assuming $\mathcal{NP} \neq \mathcal{P}$) this is also a lower bound [4]. For the p-dispersion-sum problem it is open whether an approximation algorithm with fixed ratio ρ exists, but if the triangle inequality is satisfied, an approximation algorithm with ratio $\rho = 4$ has been presented by Ravi et al. [4]. It is unknown whether this is a lower bound.

Although no approximation algorithm with fixed ratio-bound ρ have been found for the PDSP without triangle inequality assumptions, Kortsarz and Peleg [5] gave an approximation algorithm with variable approximation ratio of $O(n^{0.3885})$—which, e.g., is $\rho_{n=10} = 2.446$ and $\rho_{n=100} = 5.984$. Asahiro et al. [1] presented an approximation algorithm based on greedy removal of facilities, where the approximation ratio is bounded by

$$\begin{aligned} &\left(\frac{1}{2}+\frac{n}{2p}\right)^2 - O\left(\frac{1}{n}\right) \leqslant \rho \leqslant \left(\frac{1}{2}+\frac{n}{2p}\right)^2 + O\left(\frac{1}{n}\right) \quad \text{for } \frac{n}{3} \leqslant p \leqslant n, \\ &2\left(\frac{n}{p}-1\right) - O\left(\frac{1}{p}\right) \leqslant \rho \leqslant 2\left(\frac{n}{p}-1\right) + O\left(\frac{n}{p^2}\right) \quad \text{for } p < \frac{n}{3}. \end{aligned} \tag{1}$$

For the case $p = n/2$ the approximation ratio becomes $\rho = 9/4 \pm O(1/n)$. A different approach is to consider the case where $p = cn$ for a constant $c < 1$. In this case, Srivastav and Wolf [6] presented an approximation algorithm with ratio-bound $\rho_{c=1/2} = 2.073$, $\rho_{c=1/3} = 2.982$ and $\rho_{c=1/4} = 4.189$. Ravi et al. [4] considered other special cases, e.g. where the facilities are located in one or two dimensions of the plane and Euclidean distances are used for d_{ij}. Hassin et al. [7] consider approximation algorithms for the PDSP which satisfy the triangle inequality where k subsets of p facilities are to be selected. Corneil and Perl [8] consider the PDSP on different perfect graphs assuming that all distances are the same. Polynomial algorithms are developed for problems defined on e.g. trees and cographs while problems defined on other types of perfect graphs are shown to be $\mathcal{NP}$-hard.

In contrast to the large number of theoretical results for the two problems, not very much experimental work has been done: An exact algorithm for the *p*-dispersion problem based on branch-and-bound was presented by Erkut [3], while Kincaid [9] presented metaheuristics based on simulated annealing and tabu-search for the solution of the *p*-dispersion-sum and *p*-dispersion problem. Billionnet and Faye [10] consider lower bounds for the PDSP in minimization version. The bounds are based on a reformulation of the objective function into the sum of a constant term and a quadratic posiform. It is shown how the reformulation leading to the best lower bound by this approach can be found in polynomial time through linear programming. The approach is parallel to the technique by Hammer et al. [11] for the unconstrained quadratic 0–1 minimization problem.

The PDSP problem can be seen as a generalization of the *dense subgraph problem* (see e.g. Asahiro [12]). In this problem one considers a graph (V, E) and the objective is to select a node subset $K \subseteq V$ of cardinality $|K| = p$ such that the subgraph of G induced by K contains as many edges as possible. This problem can be modeled as a PDSP by setting $d_{ij} = 1$ iff the edge $(i, j) \in E$.

In the more general *quadratic knapsack problem* (*QKP*) each facility has an associated weight w_i and the problem is to maximize the overall distance sum between established facilities subject to an upper limit c on the applied weights. Exact algorithms for this problem include Michelon and Veuilleux [13], Hammer and Rader [14], Billionnet et al. [15], Pisinger et al. [16] and Caprara et al. [17]. The latter algorithm is based on branch-and-bound search where tight bounds are found through a reformulation. The present paper makes use of the same fundamental idea, but we are able to derive a tight reformulation in time $O(n^3)$ as opposed to the expensive subgradient optimization algorithm presented in [17]. Also, the time bounds for deriving upper bounds inside the branch-and-bound algorithm are tighter for the PDSP than for the QKP.

In the sequel we consider the most general case of *p*-dispersion problems, where the distances d_{ij} do not need to satisfy the triangle inequality and in particular they may take on positive as well as negative values. Hence any kind of *push* and *pull* constraints between the individual facilities may be modeled as described in Krarup et al. [18].

The organization of the paper is as follows: We start by considering the PDSP in Section 2. Polynomial upper bounds are presented in Section 2.1–2.3 based on reformulation, Lagrangian relaxation and semidefinite programming. The main branch-and-bound algorithm is presented in Section 2.4, and it is shown how upper bounds can be derived in $O(n)$ time inside this algorithm. Section 2.5 concludes the treatment of the PDSP by showing some computational results.

Section 3 considers the *p*-dispersion problem. The bounds presented for the PDSP are generalized to the *p*-dispersion problem, and it is discussed whether a reformulation algorithm may be applied for this problem. Finally, an exact algorithm is presented based on a reduction of the problem to a number of PDSPs. Some computational results with this algorithm are presented in Section 3.1. The paper is concluded by summing up the obtained results in Section 4. A preliminary version of the present paper appeared in [30].

2. The *p*-dispersion-sum problem

Given $N = \{1, 2, \ldots, n\}$ locations, the PDSP asks to establish p, $(1 \leqslant p \leqslant n)$ of these facilities such that the total distance sum is maximized. The distance between facility i and j is given by a rational number d_{ij}. In most facility location problems $d_{ii} = 0$ but the following discussion is not restricted to this case.

If we use the boolean variable x_j to indicate whether a facility is opened, we may formulate the PDSP as the following integer optimization problem:

$$\begin{aligned} &\text{maximize} \sum_{i\in N}\sum_{j\in N} d_{ij}x_i x_j \\ &\text{subject to} \sum_{j\in N} x_j = p \\ &\qquad x_j \in \{0,1\}, \quad j \in N. \end{aligned} \tag{2}$$

Without loss of generality, we may assume that all distances $d_{ij} \geqslant 0$, as otherwise a large constant M may be added to all values of d_{ij}. This transformation preserves the optimal solution, although the solution value gets increased by Mp^2.

2.1. Upper bounds from reformulation

An upper bound to PDSP can be found in $O(n^2)$ time by splitting the objective function into two parts. As the objective function can be written

$$\text{maximize} \sum_{j\in N}\left(\sum_{i\in N} d_{ij}x_i\right)x_j \tag{3}$$

we first derive an upper bound on the term inside the parenthesis for each value of j. Since the term $\sum_{i\in N} d_{ij}x_i$ only will contribute to the sum in (3) when $x_j = 1$ we get the following bound:

$$d'_j = \max\left\{ d_{jj} + \sum_{i\in N\setminus\{j\}} d_{ij}y_i^j : \sum_{i\in N\setminus\{j\}} y_i^j = p-1;\ y_i^j \in \{0,1\}, i \in N\setminus\{j\}\right\}. \tag{4}$$

Having derived the values d'_j for each $j \in N$, an upper bound on (2) is then derived as

$$\mathscr{U}_1 = \max\left\{ \sum_{j\in N} d'_j x_j : \sum_{j\in N} x_j = p;\ x_j \in \{0,1\},\ j \in N\right\}. \tag{5}$$

Problems (4) and (5) basically ask to choose the $p-1$ or p largest values among a set of values $d_1, \ldots, d_n$. These problems can obviously be solved in linear time through a median search algorithm. To derive the bound $\mathscr{U}_1$, we solve $n+1$ subproblems each demanding $O(n)$ time, getting an overall time bound of $O(n^2)$. Since the input size is $O(n^2)$ we derive $\mathscr{U}_1$ in linear time. An example of deriving the bound is found in Fig. 1.

A tighter upper bound can however be obtained by noting that if facility i and j are established, then we get the contribution $d_{ij} + d_{ji}$ in the objective function. As long as $d_{ij} + d_{ji}$ are unchanged, we may divide the "distances" between d_{ij} and d_{ji} as we will. Hence let (Λ) be an $n \times n$ matrix satisfying that

i \ j	1	2	3	4	5	6	7
1	0	3	7	4	10	5	7
2	3	0	9	5	5	10	6
3	7	9	0	1	3	2	4
4	4	5	1	0	1	9	1
5	10	5	3	1	0	3	2
6	5	10	2	9	3	0	3
7	7	6	4	1	2	3	0

i \ j	1	2	3	4	5	6	7
1	0	3	5	3	13	5	11
2	3	0	12	3	3	10	5
3	9	6	0	1	2	2	4
4	5	7	1	0	1	5	1
5	7	7	4	1	0	3	2
6	5	10	2	13	3	0	3
7	3	7	4	1	2	3	0

Fig. 1. Left: An instance with $n=7$ facilities and $p=3$. We find that $d'_1=17$, $d'_2=19$, $d'_3=16$, $d'_4=14$, $d'_5=15$, $d'_6=19$ and $d'_7=13$. Hence the final upper bound is $\mathscr{U}_1=55$. Right: A reformulation of the instance. Now we find that $d''_1=16$, $d''_2=17$, $d''_3=17$, $d''_4=16$, $d''_5=16$, $d''_6=15$ and $d''_7=16$. Hence the final upper bound is $\mathscr{U}_2=50$. The optimal solution is $x_2=x_4=x_6=1$ with objective value 48.

$\lambda_{ij}+\lambda_{ji}=0$ for all values of $i, j \in N$. Problem (2) is now equivalent with

$$\begin{aligned} &\text{maximize} \sum_{i\in N}\sum_{j\in N}(d_{ij}+\lambda_{ij})x_i x_j \\ &\text{subject to} \sum_{j\in N} x_j = p \\ &\qquad x_j \in \{0,1\}, \quad j \in N. \end{aligned} \tag{6}$$

Using the same bounding technique as in (5) we first derive

$$d''_j = \max\left\{ d_{jj} + \sum_{i\in N\setminus\{j\}} (d_{ij}+\lambda_{ij}) y_i^j : \sum_{i\in N\setminus\{j\}} y_i^j = p-1;\ y_i^j \in \{0,1\}, i \in N\setminus\{j\} \right\} \tag{7}$$

for each $j \in N$, which leads to the upper bound

$$\mathscr{U}_1(\Lambda) = \max\left\{ \sum_{j\in N} d''_j x_j : \sum_{j\in N} x_j = p;\ x_j \in \{0,1\},\ j \in N \right\}. \tag{8}$$

We wish to choose the matrix $\Lambda=\{\lambda_{ij}\}$ such that the tightest possible bound $\mathscr{U}_1(\Lambda)$ is derived for (6), thus we have the dual problem

$$\begin{aligned} &\text{minimize } \mathscr{U}_1(\Lambda) \\ &\text{subject to } \lambda_{ij}+\lambda_{ji}=0 \quad i, j \in N. \end{aligned} \tag{9}$$

The optimal choice of Λ can be found through linear programming. A suboptimal but considerably faster choice of Λ can however be derived through the following reformulation algorithm. We consider a sequence $\Lambda^1, \Lambda^2, \ldots, \Lambda^n$ of matrices which in each iteration attempts to tighten the bound $\mathscr{U}_1(\Lambda^k)$. Initially we set $\lambda^1_{ij}=0$ in Λ^1. Each subsequent value of Λ^{k+1} is derived from Λ^k as follows. Assume that y_i^j are the solution vectors to (7) for each value of $j \in N$, and that x_i is the solution vector to (8) for the present value of Λ^k. Then we set

$$\lambda^{k+1}_{ij} := \lambda^k_{ij} + (y_i^j x_i - y_j^i x_j)\frac{p_{ij}+p_{ji}}{2k}, \quad i, j \in N. \tag{10}$$

The motivation for this recursion is, that if $y_i^j x_i = 1$ and $y_j^i x_j = 0$ then the value of $(d_{ij} + \lambda_{ij}^k)$ contributed to the bound $\mathscr{U}_1(\Lambda^k)$ while $(d_{ji} + \lambda_{ji}^k)$ did not contribute to the bound. In order to decrease the bound, we thus attempt to decrease λ_{ij}^{k+1} and increase λ_{ji}^{k+1} in the next iteration. By having k in the denominator, we achieve smaller adjustments as k increases.

The bound $\mathscr{U}_2$ is obtained by choosing the best value of $\mathscr{U}_1(\Lambda)$ thus

$$\mathscr{U}_2 = \min_{k=1,\ldots,n} \mathscr{U}_1(\Lambda^k). \tag{11}$$

Since each value of $\mathscr{U}_1(\Lambda^k)$ can be derived in $O(n^2)$ time and each new value of Λ^k can be derived from Λ^{k-1} in $O(n^2)$ in (10), the bound $\mathscr{U}_2$ is derived in $O(n^3)$ time. As the input size is $m = n^2$ this time bound can also be written $O(m\sqrt{m})$.

Proposition 1. *The bound $\mathscr{U}_2$ is tighter than the bound $\mathscr{U}_1$ given by* (5).

Proof. Since Λ^1 has $\lambda_{ij} = 0$, we immediately observe that $\mathscr{U}_2 \leqslant \mathscr{U}_1(\Lambda^1) = \mathscr{U}_1$. The example in Fig. 1 presents a specific instance where $\mathscr{U}_2 < \mathscr{U}_1$. □

In the branch-and-bound algorithm to be described in Section 2.4, we choose the matrix Λ for which the smallest value of $\mathscr{U}_2$ was obtained, and consider $(d_{ij} + \lambda_{ij})$ as being the new distances in the reformulated problem.

The presented bound $\mathscr{U}_1$ has similarities to the bounds presented by Gallo et al. [19] for the QKP based on upper planes. The bound $\mathscr{U}_2$ is based on the reformulation scheme used by Carresi and Malucelli [20] for the quadratic assignment problem, and Caprara et al. [17] for the QKP. Where the latter used time-consuming subgradient optimization based on the Held and Karp framework [21] to derive a good reformulation, a much faster scheme is used here based on the series $1/(2k)$, $k = 1, \ldots, n$. Computational experiments with the two reformulation approaches showed that although tighter bounds could be derived at the root node by using the Held and Karp framework, the additional computational effort did not pay off with respect to the number of branch-and-bound nodes visited in the sequel. As the bound $\mathscr{U}_2$ can be derived efficiently, one may (in principle) evaluate it at every node of the branch-and-bound tree.

2.2. Upper bounds from Lagrangian relaxation

Using a similar approach as in Chaillou et al. [22] one may relax the cardinality constraint, and solve the associated Lagrangian dual problem. Using the result of Geoffrion [23], we note that the Lagrangian dual bound is equivalent to the bound obtained through continuous relaxation of PDSP. Using multiplier $\lambda \in \mathbb{R}$ we get the problem

$$\begin{aligned} &\text{maximize} \sum_{i\in N}\sum_{j\in N} d_{ij}x_i x_j - \lambda\left(\sum_{j\in N} x_j - q\right) \\ &\text{subject to } x_j \in \{0,1\}, \quad j \in N. \end{aligned} \tag{12}$$

By setting

$$\tilde{d}_{ij} := \begin{cases} d_{ij} & \text{if } i \neq j, \\ d_{ij} - \lambda & \text{if } i = j, \end{cases} \tag{13}$$

the relaxed problem can be reformulated as a *quadratic* 0-1 *programming problem*, $L_\lambda(\text{PDSP})$ of the form

$$\begin{aligned} &\text{maximize} \sum_{i\in N}\sum_{j\in N} \tilde{d}_{ij}x_i x_j + \lambda p \\ &\text{subject to } x_j \in \{0,1\}, \quad j \in N. \end{aligned} \tag{14}$$

The problem can, for a given λ, be solved efficiently as a *maximum flow problem*. The capacitated network $\mathcal{N} = (V, E, c)$ is defined with a vertex set $V = \{s, 1, \ldots, n, t\}$ where s is the source and t is the sink. The edge set E is defined on $V \times V$ having the capacities

$$c_{ij}(\lambda) := \begin{cases} \max\left(0, \sum_{k\in N} d_{jk} - \lambda\right) & \text{if } i = s,\ j = 1,\ldots,n, \\ d_{ij} & \text{if } i, j = 1,\ldots,n, \\ \max\left(0, \lambda - \sum_{k\in N} d_{ik}\right) & \text{if } j = t,\ i = 1,\ldots,n. \end{cases} \tag{15}$$

Assume that the value of the maximum flow from s to t in $\mathcal{N}$ is $\Psi(\mathcal{N})$. Then we have

Proposition 2. *An optimal solution value to* (12) *is given by*

$$z(L_\lambda(\text{PDSP})) = \lambda q + \sum_{i\in N} c_{si}(\lambda) - \Psi(\mathcal{N}). \tag{16}$$

Proof. Adapting the ideas of Chaillou et al. [22] we note that a cut $c(S, T)$ separates the set of nodes V into two subsets S and T where $s \in S$ and $t \in T$. Let $x_i = 1$ if node i is in set S and $x_i = 0$ if it is in set T. The capacity of a cut is hence $\sum_{i\in N} c_{si}(1 - x_i) + \sum_{i\in N}\sum_{j\in N} c_{ij}x_i(1 - x_j) + \sum_{i\in N} c_{it}x_i$. Since the maximum flow $\Psi(\mathcal{N})$ equals a minimum cut $c(S, T)$ we obtain

$$\Psi(\mathcal{N}) = \min_{x\in\{0,1\}^n} \left\{ \sum_{i\in N} c_{si}(1 - x_i) + \sum_{i\in N}\sum_{j\in N} c_{ij}x_i(1 - x_j) + \sum_{i\in N} c_{it}x_i \right\}.$$

As $c_{it} - c_{si} = \lambda - \sum_{j\in N} c_{ij}$ we get

$$\begin{aligned} \Psi(\mathcal{N}) &= \sum_{i\in N} c_{si} + \min_{x\in\{0,1\}^n} \left\{ \sum_{i\in N} \lambda x_i - \sum_{i\in N}\sum_{j\in N} c_{ij}x_i x_j \right\} \\ &= \sum_{i\in N} c_{si} - z(L_\lambda(\text{PDSP})) + \lambda q. \qquad \square \end{aligned} \tag{17}$$

The Lagrangian dual problem of (12) is

$$\mathcal{W}_3 = \min_{\lambda\in\mathbb{R}} L_\lambda(\text{PDSP}). \tag{18}$$

Chaillou et al. [22] proved that the Lagrangian dual is a piecewise linear function of λ with at most n linear segments. This observation can be used to construct a simple binary search algorithm for determining the optimal choice λ^* of Lagrangian multiplier, using no more than $O(n)$ iterations, each time solving a maximum flow problem on a graph with $n + 2$ vertices and $2n + n^2$ edges. Using Karzanovs algorithm [24] takes $O(|V|^3)$ time for each maximum flow problem, hence in total $O(n^4)$ time is used.

2.3. Upper bounds from semidefinite programming

A number of upper bounds for PDSP can be obtained through semidefinite programming by adapting the bounds proposed for the QKP by Helmberg et al. [25,26] to the PDSP.

The notation used in the sequel is as follows: The solution variables will be written as a column vector $x = (x_1, \ldots, x_n)^{\mathrm{T}}$. The *inner product between matrices* $A, B \in \mathbb{R}^{m\times n}$ is defined as $\langle A, B\rangle := \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}b_{ij}$. The *vector product* of two vectors $a, b \in \mathbb{R}^n$ is

$$ab^{\mathrm{T}} = \begin{pmatrix} a_1b_1 & \cdots & a_1b_n \\ \vdots & & \vdots \\ a_nb_1 & \cdots & a_nb_n \end{pmatrix}.$$

The *vector of diagonal elements* of a square matrix $A \in \mathbb{R}^{n\times n}$ will be denoted $\mathrm{diag}(A)$ and is defined as follows

$$\mathrm{diag}(A) := \mathrm{diag}\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} \\ \vdots \\ a_{nn} \end{pmatrix}. \tag{19}$$

The *rank* $\mathrm{rank}(A)$ *of a matrix* A is defined as the number of linearly independent columns of A. The *identity matrix* is denoted I, and the *unit matrix* is denoted J. A matrix $A \in \mathbb{R}^{n\times n}$ is said to be *positive semidefinite* if we have for all vectors $y \in \mathbb{R}^n$ that $y^{\mathrm{T}}Ay \geqslant 0$. We will use the notation $A \succeq 0$ to indicate that A is symmetric and positive semidefinite.

For a binary decision variable $x \in \{0, 1\}^n$ we note that the diagonal entries of $X = xx^{\mathrm{T}}$ are also binary entries $X_{ii} = x_i^2 = x_i$. Using the same modeling techniques as in Helmberg et al. [26], the PDSP may be formulated as:

$$\begin{aligned} &\text{maximize} \langle D, X\rangle \\ &\text{subject to} \langle I, X\rangle = p, \\ &\qquad X \succeq 0, \\ &\qquad \mathrm{rank}(X) = 1, \\ &\qquad X_{ii} \in \{0, 1\}. \end{aligned} \tag{20}$$

By dropping the $\mathrm{rank}(X) = 1$ constraint, and relaxing the last constraint we get a semidefinite relaxation on the form

$$\begin{aligned} &\text{maximize} \langle D, X\rangle \\ &\text{subject to} \langle I, X\rangle = p, \\ &\qquad X \succeq 0, \end{aligned} \tag{21}$$

leading to the bound $\mathscr{U}_4$.

A second bound is obtained by noting that for binary variables we have that $X \succeq 0$ and $X = xx^{\mathrm{T}}$ implies that $X - \mathrm{diag}(X)\mathrm{diag}(X)^{\mathrm{T}} \succeq 0$. This leads to the relaxation

$$\begin{aligned} &\text{maximize} \langle D, X\rangle \\ &\text{subject to} \langle I, X\rangle = p, \\ &\qquad X - \mathrm{diag}(X)\mathrm{diag}(X)^{\mathrm{T}} \succeq 0, \end{aligned} \tag{22}$$

which gives the bound $\mathscr{U}_5$. The last constraint can be written in standard form by using the Schur Complement to get

$$X - \mathrm{diag}(X)\mathrm{diag}(X)^{\mathrm{T}} \succeq 0 \quad \Leftrightarrow \quad \tilde{X} = \begin{pmatrix} 1 & \mathrm{diag}(X)^{\mathrm{T}} \\ \mathrm{diag}(X) & X \end{pmatrix} \succeq 0.$$

The next relaxation is obtained by noting that $\langle I, X\rangle = p$ implies $\langle I, X\rangle^2 = p^2$ which can be rewritten $\langle J, X\rangle = p^2$ leading to the relaxation

$$\begin{aligned} &\text{maximize} \langle D, X\rangle \\ &\text{subject to} \langle J, X\rangle = p^2, \\ &\qquad X - \mathrm{diag}(X)\mathrm{diag}(X)^{\mathrm{T}} \succeq 0, \end{aligned} \tag{23}$$

which gives the bound $\mathscr{U}_6$.

By multiplying the constraint $\sum_{i\in N} x_i = p$ by $\sum_{j\in N} x_j$ we get $\sum_{i\in N} x_i \sum_{j\in N} x_j = p\sum_{j\in N} x_j$, which can be rewritten as $\sum_{j\in N}\left(-p + \sum_{i\in N} x_i\right) x_j = 0$. This leads to the following relaxation:

$$\begin{aligned} &\text{maximize} \langle D, X\rangle \\ &\text{subject to} \left\langle \begin{pmatrix} p \\ -\mathrm{diag}(I) \end{pmatrix} \begin{pmatrix} 0 \\ \mathrm{diag}(I) \end{pmatrix}^{\mathrm{T}}, \tilde{X} \right\rangle \succeq 0 \\ &\qquad X - \mathrm{diag}(X)\mathrm{diag}(X)^{\mathrm{T}} \succeq 0, \end{aligned} \tag{24}$$

which gives the bound $\mathscr{U}_7$.

Finally, we may multiply the cardinality constraint with each of the variables x_i for $i \in N$ getting equations $\sum_{j\in N} x_i x_j = x_i p$ for $i \in N$. By introducing X_{ij} for $x_i x_j$ and X_{ii} for x_i we get the equations $\sum_{j\in N} X_{ij} = X_{ii} p$ for $i \in N$, which leads to the following relaxation:

$$\begin{aligned} &\text{maximize} \langle D, X\rangle \\ &\text{subject to} \sum_{j\in N} X_{ij} = pX_{ii}, \quad i \in N, \\ &\qquad X - \mathrm{diag}(X)\mathrm{diag}(X)^{\mathrm{T}} \succeq 0, \end{aligned} \tag{25}$$

giving the bound $\mathscr{U}_8$.

2.4. The main branch-and-bound algorithms

Our branch-and-bound algorithm makes use of the upper bound $\mathscr{U}_2$ described in Section 2.1 as experimental results in [27] for the QKP have shown that this bound is quite tight and still computationally cheap to derive.

At the root node of the branching tree, we apply the reformulation algorithm (10) with an embedded heuristic procedure so as to define tight upper and lower bounds, as well as a convenient problem reformulation. The reformulation algorithm is followed by a reduction procedure in which we try to fix some variables at their optimal value.

The nodes of the branching tree other than the root are processed as fast as possible, thus no heuristic, reduction, or updating of the Λ matrix are performed. We simply derive the $\mathscr{U}_2$ bound (associated with

the best Λ matrix found at the root node), in linear time, possibly update the incumbent solution and then branch on the first free variable x_i. The details of the branch-and-bound algorithm are outlined in the following sections.

2.4.1. Heuristics

In order to derive a good initial solution, we implemented the heuristic algorithm for QKP devised by Billionnet and Calmels [28], and then used by Caprara et al. [17]. This algorithm is based on the greedy principle, where first an initial solution is obtained by setting $x_j = 1$ for all $j \in N$, and then iteratively choosing the variable i which can be changed from $x_i = 1$ to $x_i = 0$ with minimal loss in the objective function. This sequence is repeated until $\sum_{j \in N} x_j = p$.

The second part of the algorithm is an ordinary 2-opt algorithm which repeatedly selects two variables with $x_i = 1$ and $x_j = 0$ such that if their values are exchanged, the largest increase in the objective function is achieved.

We apply the heuristic in its full form at the first step of our algorithm. Additionally, at each step of the reformulation algorithm (10) we apply the 2-opt algorithm for the present solution vector x to (8). As will be shown in the computational experiments, this approach gives very good initial solutions.

It is worth noting that the first step of the algorithm is identical to the greedy algorithm presented by Asahiro et al. [1], thus the approximation ratio (1) holds for our initial heuristic.

2.4.2. Reduction

In order to reduce the size of an instance, we apply a simple reduction algorithm immediately after the reformulation algorithm: Assume that we have an incumbent solution x with objective value z. Let $\mathscr{U}_2^{x_i=0}$ be an upper bound on (2) with the additional constraint that $x_i = 0$. If $\mathscr{U}_2^{x_i=0} \leqslant z$ we may conclude that $x_i = 1$ in every improved solution, and thus fix x_i to 1 during the remaining algorithm. In a similar way we let $\mathscr{U}_2^{x_i=1}$ be an upper bound on (2) with additional constraint that $x_i = 1$. If $\mathscr{U}_2^{x_i=1} \leqslant z$ we may fix x_i to 0.

The time complexity of the reduction algorithm is $O(n^3)$ since for each variable x_i we derive the bound $\mathscr{U}_2$ in $O(n^2)$ time. Notice that we use the same value of Λ for all reductions even though a better reformulation may exist when imposing an additional constraint on x_i. Having fixed a variable x_i to 0, we remove the corresponding row i and column i from the distance matrix and decrease n. This also happens if the variable x_i is fixed to 1, but in this case we also decrease p and set $d_{jj} := d_{jj} + d_{ij} + d_{ji}$ for each $j \neq i$. In addition, a constant term of d_{ii} should be added to the objective function.

2.4.3. Branching scheme

The branching scheme is based on a simple depth-first approach, where we at each node derive an upper bound for the free variables based on the $\mathscr{U}_2$ bound (11). Since it is very time consuming to adjust the multipliers Λ at every node, we chose to use the multipliers derived at the root node throughout the whole search tree. This obviously means that we do not obtain the tightest possible bounds, but on the other hand we can derive the bounds in $O(n)$ as will be shown in the next section.

The order of the branching variables is fixed in advance. For each variable i we derive the value

$$D_i = \max \left\{ d_{ii} + \sum_{j \in N \setminus \{i\}} (d_{ij} + d_{ji}) y_j^i : \sum_{j \in N \setminus \{i\}} y_j^i = p - 1;\ y_j^i \in \{0, 1\},\ j \in N \setminus \{i\} \right\}. \tag{26}$$

i \ j	1	2	3	4	5	6	7
1	9	10	12	13	13	10	11
2	**7**	**7**	**5**	**3**	**3**	**5**	**5**
3	5	7	4	3	3	5	4
4	5	7	4	1	2	3	3
5	3	6	2	1	2	3	2
6	3	3	1	1	1	2	1
7	0	0	0	0	0	0	0
d''_j	16	17	17	16	16	15	16

i \ j	1	2	3	4	5	6	7
1	9	10	12	13	13	10	11
2	7	**7**	5	**3**	3	**5**	5
3	5	7	**4**	3	**3**	5	**4**
4	5	7	4	1	2	3	3
5	3	6	2	1	2	3	2
6	3	3	1	1	1	2	1
7	0	0	0	0	0	0	0
d''_j	—	17	16	16	6	15	9

Fig. 2. Left: The table from Fig. 1 (reformulated problem) with each column ordered according to nonincreasing distances. The bold numbers indicate the $(p-1)$th value in each column, and the button line shows the current values of d''_j. Right: Having set $x_1 = 0$ the first column disappears as well as several entries in the other rows. This affects the position of the $(p-1)$th value in each column as well as all values of d''_j.

The values D_i are a kind of estimate on the expected contribution of facility i to the objective value, and thus we order the variables such that those variables having the largest value of D_i are considered first. The values D_i are similar to the d''_j values given by (7) but it turned out that the D_i values are more appropriate as a guideline for the branching process. As seen from Fig. 1 (right) the reformulation attempts to smooth out the values of d''_j and thus leave no information for choosing the branching order.

The branch-and-bound algorithm is a recursive algorithm which in each step fixes the first free variable x_i to 1 and then 0. Having fixed the variable to $x_i = 1$ we decrease p, add the term $d_i i$ to the objective value, set $d_{jj} := d_{jj} + d_{ij} + d_{ji}$ for $j = i+1, \ldots, n$, and finally remove the row i and column i from the problem. If the variable is fixed to $x_i = 0$ we only need to remove the row and column i from the problem.

2.4.4. Fast upper bounds

The inner loop of the branch-and-bound algorithm can be performed in $O(n)$ time, while a direct evaluation of the bound $\mathscr{U}_2$ according to (7) and (8) demands $O(n^2)$. By using the information from one branching node to the next, we may however decrease the time for deriving bounds to $O(n)$ as follows:

At the root node, we sort each column of the distance matrix $\{d_{ij}\}$ according to nonincreasing values (see Fig. 2), and use a double-linked list to store the values. Additional information is saved, so that one can find from an entry in the original distance matrix to the corresponding entry in the sorted distance matrix. A pointer to the $(p-1)$th element in each column is maintained together with the current sum of the first p elements.

If we consider a single column j of the sorted distance matrix during the branch-and-bound algorithm, then two changes can happen to the list. If a variable x_i is fixed to 0, then the corresponding distance d_{ij} in the sorted matrix will be removed from the column (which can be done in constant time since we have double-linked lists), and we may need to move our pointer to the $(p-1)$th element one step forward. If the variable x_i on the other hand is fixed to 1, then the distance d_{ij} is removed from the column and we decrease $(p-1)$ by one, which may imply that the pointer to the $(p-1)$th element is moved one step backward. Knowing the old value of d''_j it is easy to derive the new value by only adding or subtracting the performed changes. Since both of the operations can be done in constant time, we can determine d''_j for all $j \in N$ in $O(n)$ time. The final bound $\mathscr{U}_2$ is thus also derived in $O(n)$ time.

2.5. Computational experiments

To evaluate the practical performance of the algorithm, we ran a number of computational experiments on instances with specific properties. For this purpose, we considered the following classes of randomly generated instances:

GEO: The *geometrical problems* reflects a typical location problem in the Euclidean space, as presented in Erkut [3]. The n facilities are randomly located in a 100×100 rectangle, and d_{ij} is the Euclidean distance between facilities i and j.

WGEO: The *weighted geometrical problems* are constructed to illustrate the case where the facilities have different weights (e.g. the radio transmitters located at some positions have different effect). As previously, the n facilities are randomly located in a 100×100 rectangle, and each facility is assigned a weight w_i in the interval $[5 \ldots 10]$. The distance d_{ij} is then found as $w_i w_j$ times the Euclidean distance between facility i and j.

EXP: Instances with *exponential distribution* of the distances were presented in Kincaid [9] (class 2). These instances should investigate how the algorithms perform when the triangle inequality was not satisfied. Each d_{ij} with $i < j$ is randomly drawn from an exponential distribution with mean value 50. We set $d_{ij} = d_{ji}$ to ensure symmetry.

AEXP: An *asymmetric exponential distribution* is also considered in Kincaid [9] (class 3). These instances should reflect the case when neither triangle inequality or symmetry is satisfied.Each d_{ij} with $i \neq j$ is randomly drawn from an exponential distribution with mean value 50.

RAN: Instances with *random distances* are generated with d_{ij} randomly distributed in $[1 \ldots 100]$.

DSUB: Finally *dense subgraph* instances reflect the unweighted case. Hence d_{ij} is set to 100 or 0 with 50% probability each.

In all the instances, we set $d_{ii} = 0$ for $i = 1, \ldots, n$. The number of facilities p is in each instance randomly chosen in the interval $[2 \ldots n-2]$. The following results are average values of 10 instances, and solution times are reported for a AMD 64-bit, 2.4 GHz. A time limit of 15 h was assigned to each series of 10 instances, and a dash indicates when the optimal solution could not be found within this limit.

First, Table 1 presents the quality of the upper bound $\mathscr{U}_2$. The deviation in percent is derived as $100(\mathscr{U}_2 - z^*)/z^*$, where z^* is the optimal solution value. It is seen that the upper bound is reasonably tight for most of the instances, and becomes as low as 2% for large-sized weighted geometrical problems.

Table 2 gives the absolute deviation of the heuristic solution performed at the root node before the reformulation algorithm was run. It is seen, that the heuristic is able to find very good initial solutions. Even better heuristic solutions were found by the improvement algorithm performed as part of the reformulation scheme (i.e. the solution of (8) followed by a 2-opt algorithm) as seen in Table 3. Indeed the optimal solution is found by the heuristic in all the instances considered!

Table 4 gives the number of variables fixed at their optimal value by the reduction algorithm. It is seen, that on average only a few variables can be fixed. The gap between the upper and lower bound must be below a few percent for the reduction algorithm to work appropriately.

Finally the solution times for solving the problems to optimality are given in Table 5. The geometric problems tend to be easier to solve, since we solve Euclidean geometric problems up to $n = 90$ and weighted geometric problems up to $n = 250$. The remaining problems are somehow more difficult to solve, but still instances up to $n = 60$ are solved. For comparison it should be mentioned that Kincaid [9],

Table 1
Relative deviation of upper bound $\mathscr{U}_2$ in pct

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	7.66	6.76	7.90	7.57	7.68	8.30
20	9.04	4.03	11.11	11.60	11.94	16.37
30	9.15	3.25	14.84	13.40	12.88	16.58
40	8.59	1.47	14.97	13.45	8.64	11.24
50	9.06	3.39	16.17	9.77	20.88	27.86
60	19.68	2.68	22.32	15.69	15.52	23.88
70	10.79	3.20	17.08	—	—	—
80	6.93	2.76	—	—	—	—
90	8.17	2.92	—	—	—	—
100	—	7.10	—	—	—	—
150	—	2.38	—	—	—	—
200	—	2.78	—	—	—	—
250	—	2.20	—	—	—	—

Average of 10 instances.

Table 2
Relative deviation of initial heuristic solution in pct

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	0.0	0.2	2.0	1.2	0.5	0.0
20	0.0	0.0	0.8	0.8	0.4	0.7
30	0.0	0.0	0.8	0.3	0.4	0.4
40	0.0	0.0	0.6	0.3	0.2	0.3
50	0.0	0.0	0.4	0.5	0.3	0.2
60	0.0	0.0	0.4	0.9	0.3	0.4
70	0.0	0.0	0.4	—	—	—
80	0.0	0.0	—	—	—	—
90	0.0	0.0	—	—	—	—
100	—	0.0	—	—	—	—
150	—	0.0	—	—	—	—
200	—	0.0	—	—	—	—
250	—	0.0	—	—	—	—

Average of 10 instances.

using metaheuristics, obtained solutions for instances with $n = 25$ and 33 for the classes EXP and AEXP. Billionnet and Faye [10], using CPLEX to solve the problem to optimality, report results on randomly generated instances with n up to 30.

3. The p-dispersion problem

The PDSP wishes to maximize the average distance between any pair of facilities. There are however situations where the average measure is not appropriate. For instance, a shop owner would not be happy

Table 3
Relative deviation of heuristic solution in pct

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	0.0	0.0	0.0	0.0	0.0
30	0.0	0.0	0.0	0.0	0.0	0.0
40	0.0	0.0	0.0	0.0	0.0	0.0
50	0.0	0.0	0.0	0.0	0.0	0.0
60	0.0	0.0	0.0	0.0	0.0	0.0
70	0.0	0.0	0.0	—	—	—
80	0.0	0.0	—	—	—	—
90	0.0	0.0	—	—	—	—
100	—	0.0	—	—	—	—
150	—	0.0	—	—	—	—
200	—	0.0	—	—	—	—
250	—	0.0	—	—	—	—

Average of 10 instances.

Table 4
Number of variables fixed at their optimal values

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	3	5	5	5	5	8
20	2	8	1	2	1	5
30	3	9	2	3	2	4
40	4	25	5	0	0	2
50	0	16	0	2	12	20
60	0	16	9	6	2	18
70	0	14	0	—	—	—
80	0	20	—	—	—	—
90	0	13	—	—	—	—
100	—	3	—	—	—	—
150	—	27	—	—	—	—
200	—	18	—	—	—	—
250	—	38	—	—	—	—

Average of 10 instances.

to have a competing shop located next door, although this would lead to the overall best distribution of the p shops. In this case it may be appropriate to consider the *p-dispersion problem* (PDP) where the objective is to maximize the minimum distance between any two established facilities.

To be more formal, assume that $N = \{1, 2, \ldots, n\}$ facilities are given and p, $(1 \leqslant p \leqslant n)$ of these should be opened. The distance between facility i and j is given by an integer d_{ij}, where we in this case assume that $d_{ii} = \infty$. Without loss of generality, we may assume that all distances $d_{ij} \geqslant 0$, as otherwise a large constant M may be added to all values of d_{ij}. This will affect the optimal solution value by M but not the optimal solution vector.

Table 5
Solution times in seconds

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00
30	0.00	0.00	0.00	0.00	0.01	0.01
40	0.02	0.00	0.05	0.13	0.25	0.61
50	0.28	0.01	1.23	0.26	5.63	12.04
60	1.24	0.02	38.65	27.45	2001.53	4269.02
70	16.22	0.04	2906.07	—	—	—
80	60.54	0.07	—	—	—	—
90	3957.20	0.12	—	—	—	—
100	—	0.17	—	—	—	—
150	—	1.24	—	—	—	—
200	—	68.02	—	—	—	—
250	—	5132.53	—	—	—	—

Average of 10 instances.

If we use the boolean variable x_j to indicate when a facility is opened the *p-dispersion problem* may be formulated as follows:

$$\begin{aligned} &\text{maximize} && r \\ &\text{subject to} && r x_i x_j \leqslant d_{ij} \quad i, j \in N \\ &&& \sum_{j \in N} x_j = p \\ &&& x_j \in \{0, 1\}, \quad j \in N \\ &&& r \geqslant 0. \end{aligned} \tag{27}$$

The first constraint in (27) has the effect that if $x_i x_j = 1$ then $d_{ij} \geqslant r$. If on the other hand $x_i x_j = 0$ the constraint says $d_{ij} \geqslant 0$ which is satisfied by assumption.

An upper bound for the PDP can be found by using the same techniques as in the $\mathscr{U}_1$ bound for PDSP. For each $j \in N$ derive

$$r'_j = \max \left\{ r : r y_i^j \leqslant d_{ij}, i \in N \backslash \{j\}; \sum_{i \in N \backslash \{j\}} y_i^j = p - 1; y_i^j \in \{0, 1\}, i \in N \backslash \{j\}; r \geqslant 0 \right\}. \tag{28}$$

Here r'_j is an upper bound on the minimum distance from j of $p - 1$ facilities. The value r'_j is found by establishing facility j and with it the $p - 1$ facilities located furthest away from j.

Then an upper bound $\mathscr{U}_{\text{pdp}}$ on PDP is found as the solution value to

$$\mathscr{U}_{\text{pdp}} = \max \left\{ r : r x_j \leqslant r'_j, j \in N; \sum_{j \in N} x_j = p; x_j \in \{0, 1\}, j \in N; r \geqslant 0 \right\}. \tag{29}$$

Problems (28) and (29) ask to find the p-median of a set, which obviously can be done in linear time. Since we solve $n + 1$ such problems, the bound $\mathscr{U}_{\text{pdp}}$ can be derived in $O(n^2)$ time.

Proposition 3. *The value* $\mathscr{U}_{\text{pdp}}$ *gives an upper bound on* (27).

Proof. We prove the statement by showing that any feasible solution to (27) is also a feasible solution to (29). Hence assume that the pair (x, r) is a feasible solution to (27). We construct a solution to (28) and (29) for the same value of r. By setting $y_i^j = x_j$ and $r'_j = r$ for $j \in N$ we notice that (y^j, r'_j) is a feasible solution to (28). Hence the pair (x, r) is a solution to (29).

Since the values r'_j and r in (28) and (29) are found as a maximum over larger set than the original solution space we will not obtain smaller values than the optimal solution. □

It is however not obvious how we should tighten the bound $\mathscr{U}_{\text{pdp}}$. Using the same approach as for the PDSP, we may add a matrix Λ to the given distance matrix. The values λ_{ij} must however satisfy that $\lambda_{ij} = 0$ or $\lambda_{ji} = 0$ for all values of $i, j \in N$. But any choice of Λ with this property does not improve the bound $\mathscr{U}_{\text{pdp}}$.

A different approach is to decompose the p-dispersion problem (27) into a number of clique problems. Hence for a predefined value of r, define a graph $G(r) = (V, E)$ where V is the set N of locations, and $(i, j) \in E$ if and only if $d_{ij} \geqslant r$. The problem is now to find a clique of size p. If such a clique exists, we know that the current value of r is a feasible solution to (27). Since r can only attain one of the $O(n^2)$ different values in $\{d_{11}, d_{12}, \ldots, d_{nn}\}$, we may preorder these values according to increasing values and use binary search to identify the maximum value of r. Hence assuming that $\{d^1, \ldots, d^k\}$ is the ordered set of distinct values r can attain, then we look for a clique of size p in $G(d^{k/2})$. If such a clique exists, then we search for r among that $\{d^{k/2}, \ldots, d^k\}$, otherwise we search for r among the values $\{d^1, \ldots, d^{k/2-1}\}$. This process is repeated $O(\log n^2)$ times.

Answering the question whether a clique of size p exists in $G(r)$ can be restated as a dense subgraph problem. Let $e_{ij} = 1$ if and only if $(i, j) \in E$. Then the dense subgraph problem can be formulated as

$$\begin{aligned} &\text{maximize} \sum_{i \in N} \sum_{j \in N} e_{ij} x_i x_j \\ &\text{subject to} \sum_{j \in N} x_j = p \\ &\qquad x_j \in \{0, 1\}, \quad j \in N. \end{aligned} \tag{30}$$

If a solution of value $z^* = p(p-1)$ is found to (30) we know that a clique of size p exists in the graph (V, E), and thus the current value of r is feasible for (27). Fig. 3 illustrates the bound $\mathscr{U}_{\text{pdp}}$ and the reduction to a clique problem. As the dense subgraph problem (30) is a special case of the PDSP, we may use the algorithm from Section 2 to solve the problem.

3.1. Computational experiments

We conclude this section by showing some computational experiments with the presented algorithm for PDP. The instances are generated as in Section 2.5 with the only exception that d_{ii} always is set to ∞. Preliminary experiments indicated that bound $\mathscr{U}_{\text{pdp}}$ was too weak to efficiently prune the search tree of a branch-and-bound algorithm. Hence the generated problems are solved by considering a number of dense subgraph problems (30). Each dense subgraph problem is solved by use of the PDSP algorithm from Section 2.

i \ j	1	2	3	4	5	6	7
1	∞	3	7	4	10	5	7
2	3	∞	9	5	5	10	6
3	7	9	∞	1	3	2	4
4	4	5	1	∞	1	9	1
5	10	5	3	1	∞	3	2
6	5	10	2	9	3	∞	3
7	7	6	4	1	2	3	∞

i \ j	1	2	3	4	5	6	7
1	0	0	1	0	1	0	1
2	0	0	1	0	0	1	0
3	1	1	0	0	0	0	0
4	0	0	0	0	0	1	0
5	1	0	0	0	0	0	0
6	0	1	0	1	0	0	0
7	1	0	0	0	0	0	0

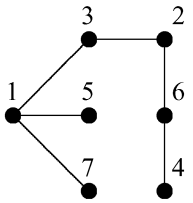

Fig. 3. Left: An instance with $n = 7$ facilities and $p = 3$. We find that $r'_1 = 7$, $r'_2 = 9$, $r'_3 = 7$, $r'_4 = 5$, $r'_5 = 5$, $r'_6 = 9$ and $r'_7 = 6$. Hence an upper bound is $\mathscr{U}_{\text{pdp}} = 7$. Right: For $r = 7$ we get the clique problem represented by the given matrix. Since a clique of size 3 does not exist, $r = 7$ is not a feasible solution value. The optimal solution is obtained with $x_2 = x_4 = x_6 = 1$ having objective value 5.

Table 6
Solution times in seconds for solving the p-dispersion problem

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.01	0.01	0.00	0.00	0.00
30	0.02	0.03	0.02	0.02	0.02	0.00
40	0.24	0.13	0.06	0.07	0.05	0.01
50	2.05	2.15	0.22	0.08	0.49	0.01
60	15.41	14.04	1.53	0.59	6.10	0.03
70	650.55	398.80	17.83	12.31	28.47	0.05
80	—	—	1151.43	11.46	314.15	0.08
100	—	—	—	—	—	0.17
200	—	—	—	—	—	2.73
300	—	—	—	—	—	22.34
400	—	—	—	—	—	50.38

Average values of 10 instances.

Table 6 gives the overall solution time as average of 10 randomly generated instances run on a AMD 64-bit, 2.4 GHz. It is interesting to note, that the two geometrical problems are slightly more difficult to solve in the PDP version than in the PDSP version. On the other hand, the non-geometric problems tend to be easier to solve in the PDP version.

For comparison, Erkut [3] reports that his branch-and-bound algorithm uses about 14 s on average for instances of size $n = 30$, while instances with $n = 40$ take half an hour to solve. The instances considered by Erkut are of the GEO type. Although Erkut performed his experiments on a PC-AT, it should be obvious that the transformation of PDP to PDSP leads to significantly better solution times.

4. Conclusion

The p-dispersion-sum problem has several applications in theory and practice, but currently no approximation algorithm with fixed ratio bound is known for its solution. For small instances, or when the

number of facilities p is a large fraction of n, one may obtain an approximation ratio of about two, which however may be insufficient in practical applications.

The study of exact algorithms gives an interesting contrast to the results on approximation algorithms. In particular it is interesting to see that moderately large problems can be solved to optimality. Knowing the exact solutions makes it possible to evaluate the quality of heuristics, and it is surprising to note that the here presented heuristic is able to find optimal solutions in all the considered instances. This indicates that although the p-dispersion-sum problem is difficult to approximate in the worst-case situation, greedy heuristics may perform very well on instances occurring in practice.

The promising solution times for the exact algorithm have been obtained by deriving tight upper bounds based on a reformulation of the problem. During the branch-and-bound algorithm these bounds can be derived in $O(n)$ time by using appropriate data structures. An algorithm for finding a near-optimal reformulation of the problem in $O(n^3)$ time was presented.

We have finally made some preliminary experiments with exact solution of the p-dispersion problem, based on reduction to a number of clique problems. The computational results show that this may be a promising way of solving the problem, although better solution times can be obtained by using more sophisticated algorithms [29] for solving the clique problems.

References

[1] Asahiro Y, Iwama K, Tamaki H, Tokuyama T. Greedily finding a dense subgraph. In: Karlsson RG, Lingas A, editors. Algorithm theory—SWAT '96, 5th Scandinavian Workshop on Algorithm Theory, Reykjavik, Iceland, July, vol. 1097. Berlin: Springer; 1996. p. 136–48.

[2] Hansen P, Moon ID. Dispersing facilities on a network. Presentation at the TIMS/ORSA Joint National Meeting, Washington, D.C.; 1988.

[3] Erkut E. The discrete p-dispersion problem. European Journal of Operational Research 1990;46:48–60.

[4] Ravi SS, Rosenkrantz DJ, Tayi GK. Heuristic and special case algorithms for dispersion problems. Operations Research 1994;42:299–310.

[5] Kortsarz G, Peleg D. On choosing a dense subgraph. In: Proceedings of the 34th Annual IEEE Symposium on Foundation of Computer Science. New York: Institute of Electrical and Electronics Engineers; 1993. p. 692–701.

[6] Srivastav A, Wolf K. Finding dense subgraphs with semidefinite programming. In: Jansen K, Rolim J, editors. Approximation algorithms for combinatorial optimization. Lecture Notes in Computer Science, vol. 1444. Berlin: Springer; 1998. p. 181–91.

[7] Hassin R, Rubinstein S, Tamir A. Approximation algorithms for maximum dispersion. Operations Research Letters 1997;21:133–7.

[8] Corneil DG, Perl Y. Clustering and domination in perfect graphs. Discrete Applied Mathematics 1984;9:27–40.

[9] Kincaid RK. Good solutions to discrete noxious location problems via metaheuristics. Annals of Operations Research 1992;40:265–81.

[10] Billionnet A, Faye A. A lower bound for a constrained quadratic 0–1 minimization problem. Discrete Applied Mathematics 1997;74:135–46.

[11] Hammer PL, Hansen P, Simeone B. Roof duality, complementation and persistency in quadratic 0–1 optimization. Mathematical Programming 1984;28:121–55.

[12] Asahiro Y, Hassin R, Iwama K. Complexity of finding dense subgraphs. Discrete Applied Mathematics 2002;121(1–3): 15–26.

[13] Michelon P, Veilleux L. Lagrangean methods for the 0–1 quadratic knapsack problem. European Journal of Operational Research 1996;92:326–41.

[14] Hammer PL, Rader Jr. DJ. Efficient methods for solving quadratic 0–1 knapsack problems. INFOR 1997;35:170–82.

[15] Billionnet A, Faye A, Soutif E. A new upper bound for the 0–1 quadratic knapsack problem. European Journal of Operational Research 1999;112:664–72.

[16] Pisinger D, Rasmussen AB, Sandvik R. Solution of large-sized quadratic knapsack problems through aggressive reduction. INFORMS Journal on Computing, 2005, to appear.
[17] Caprara A, Pisinger D, Toth P. Exact solution of the quadratic knapsack problem. INFORMS Journal on Computing 1999;11:125–37.
[18] Krarup J, Pisinger D, Plastria F. Discrete location problems with push–pull objectives. Discrete Applied Mathematics 2002;123:363–78.
[19] Gallo G, Hammer PL, Simeone B. Quadratic knapsack problems. Mathematical Programming 1980;12:132–49.
[20] Carraresi P, Malucelli F. A reformulation scheme and new lower bounds for the qap. In: Pardalos PM, Wolkowicz H., editors. Quadratic assignment and related problems. Providence, RI: American Mathematical Society Press; 1994. p. 147–60.
[21] Held M, Karp RM. The traveling salesman problem and minimum spanning trees: Part ii. Mathematical Programming 1971;1:6–25.
[22] Chaillou P, Hansen P, Mahieu Y. Best network flow bound for the quadratic knapsack problem. In: Simeone B, editor. Combinatorial optimization. Lecture Notes in Mathematics, vol. 1403. Berlin: Springer; 1986. p. 225–35.
[23] Geoffrion AM. Lagrangian relaxation for integer programming. Mathematical Programming Study 1974;2:82–114.
[24] Karzanov AV. Determining the maximal flow in a network by the method of preflow. Soviet Mathematics Doklady 1974;15:434–7.
[25] Helmberg C, Rendl F, Weismantel R. Quadratic knapsack relaxations using cutting planes and semidefinite programming. Lecture Notes in Computer Science, vol. 1084, 1996. p. 175–89.
[26] Helmberg C, Rendl F, Weismantel R. A semidefinite programming approach to the quadratic knapsack problem. Journal of Combinatorial Optimization 2000;4:197–215.
[27] Kellerer H, Pferschy U, Pisinger D. Knapsck problems. Berlin: Springer; 2004.
[28] Billionnet A, Calmels F. Linear programming for the 0–1 quadratic knapsack problem. European Journal of Operational Research 1996;92:310–25.
[29] Johnson DS, Trick MA, editors. Cliques, coloring and satisfiability: 2nd DIMACS implementation challenge. DIMACS Series in: Discrete Mathematics and Theoretical Computer Science. Providence, RI: American Mathematical Society Press; 1996.
[30] Pisinger D. Exact Solution of p-dispersion problems, Technical Report 99/14, DIKU, University of Copenhagen, Denmark; 1999.