Exact solution of p-dispersion problems*

David Pisinger
DIKU, University of Copenhagen, DK-2100 Copenhagen

December 1999

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).

Simple upper bounds for the problem are presented, and it is shown how these bounds can be tightened through a reformulation scheme which runs in $O(n^3)$ time. A branch-and-bound algorithm is then derived, which at each branching node is able to derive the upper bounds in $O(n)$ time. Computational experiments show that the algorithm may solve geometric problems of size up to $n = 80$, and weighted geometric problems of size $n = 200$.

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. Formulations and simple 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.

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* the objective is to maximize the distance sum between the p established facilities. Since the number of selected facilities is constant maximizing the distance sum is equivalent to maximizing the average distance between facilities. A different variant

*Tech. Rep. 99/14, DIKU, University of Copenhagen, Denmark

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 in locating branches of a large chain. In telecommunication one may e.g. 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 defence, since it is common practice to scatter ones installations in order to make it more difficult to the enemy to disarm them. The p-dispersion-sum problem is thus also known as the *p-defence-sum problem.* In graph theory, the *heaviest subgraph* problem considers a weighted graph (V, E, d). The problems 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 p-dispersion-sum problem.

Both of the problems are $\mathcal{NP}$-hard which easily can be proved by reduction from the clique problem [4, 3]. Even if the distance matrix satisfies the triangle inequality, the problems remain $\mathcal{NP}$-hard [4, 3]. Ravi, Rosenkrantz and Tayi [9] 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 [9]. 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 [9]. It is unknown whether this is a lower bound.

Although no approximation algorithm with fixed ratio-bound ρ have been found for the p-dispersion-sum problem, Kortsarz and Peleg [7] gave an approximation algorithm with variable approximation ratio of $O(n^{0.3885})$ — which e.g. is $\rho_{n=10} = 2.446$ and $\rho_{n=100} = 5.984$. A different approach is to consider the case where $p = cn$ for a constant $c < 1$. In this case, Srivastav and Wolf [10] 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, Rosenkrantz, Tayi [9] 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}.

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 [6] presented metaheuristics based on simulated annealing and tabu-search for the solution of the p-dispersion-sum and p-dispersion problem.

The p-dispersion-sum problem can be seen as a generalization of the *dense subgraph problem.* 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 modelled as a p-dispersion-sum problem by settting $d_{ij} = 1$ 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. Caprara, Pisinger, Toth [2] prsented an exact

algorithm for this problem based on branch-and-bound where tight bounds are found through a reformulation. The present paper relies in a large extent on the techniques developed for the QKP. In the present paper, we are however able to derive a reformulation scheme which runs in polynomial time $O(n^3)$ as opposed to the subgradient optimization algorithm presented in [2]. Also, the time bounds for deriving upper bounds are tighter for the p-dispersion-sum problem 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 modelled as described in Krarup et. al. [8]. The organization of the paper is as follows: We start by considering the p-dispersion-sum problem in Section 2. Simple upper bounds are presented, and it is shown how these bounds may be tightened through a reformulation of the problem. The main branch-and-bound algorithm is presented in 2.1, and it is shown how upper bounds can be derived in $O(n)$ time inside this algorithm. Section 2.2 concludes the treatment of the p-dispersion-sum problem by showing some computational results. Section 3 considers the p-dispersion problem. The bounds presented for the p-dispersion-sum problem 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 transformation of the problem to a number of p-dispersion-sum problems. Some computational results with this algorithm are presented in Section 3.1. The paper is concluded by summing up some of the obtained results in Section 4.

2 The p-dispersion-sum problem

Given $N = \{1, 2, \ldots, n\}$ locations, the p-dispersion-sum problem asks to establish p, $(1 \leq p \leq n)$ of these facilities such that the total distance sum is maximized. The distance between facility i and j is given by an integer d_{ij}, where we assume that $d_{ii} = 0$. If we use the boolean variable x_j to indicate whether a facility is opended, we may formulate the *p-dispersion-sum problem (PDSP)* as the following integer optimization problem:

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

Without loss of generality, we may assume that all distances $d_{ij} \geq 0$, as otherwise a large constant M may be added to all values of d_{ij}, $i \neq j$. This transformation preserves the optimal solution, although the solution value gets increased by $Mp(p-1)$.

An upper bound to PDSP can be found in $O(n^2)$ time by splitting the objective function

$_i{}^j$	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

Figure 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 $u_1 = 55$. 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 $u_2 = 50$. The optimal solution is $x_2 = x_4 = x_6 = 1$ with objective value 48.

into two parts. As the objective function can be written

$$\text{maximize} \quad \sum_{j \in N} \left(\sum_{i \in N} d_{ij} x_i \right) x_j \tag{2}$$

we first derive an upper bound on the term inside the paranthesis for each value of j. Since the term $\sum_{i \in N} d_{ij} x_i$ only will contribute to the sum in (2) when $x_j = 1$ we get the following bound:

$$d'_j = \max \left\{ d_{jj} + \sum_{i \in N \setminus \{j\}} d_{ij} y_i^j : \sum_{i \in N \setminus \{j\}} y_i^j = p - 1;\ y_i^j \in \{0,1\}, i \in N \setminus \{j\} \right\} \tag{3}$$

Having derived the values d'_j for each $j \in N$, an upper bound on (1) is then derived as

$$u_1 = \max \left\{ \sum_{j \in N} d'_j x_j : \sum_{j \in N} x_j = p;\ x_j \in \{0,1\}, j \in N \right\} \tag{4}$$

The problems (3) and (4) 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 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 u_1 in linear time. An example of deriving the bound is found in Figure 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 $\lambda_{ij} + \lambda_{ji} = 0$ for all values of $i, j \in N$. Problem (1) is now equivalent

with

$$\begin{aligned} &\text{maximize} && \sum_{i\in N}\sum_{j\in N}(d_{ij}+\lambda_{ij})x_ix_j \\ &\text{subject to} && \sum_{j\in N}x_j = p \\ & && x_j\in\{0,1\}, \qquad\qquad j\in N \end{aligned} \tag{5}$$

Using the same bounding technique as in (4) we first derive

$$d_j'' = \max\left\{d_{jj}+\sum_{i\in N\backslash\{j\}}(d_{ij}+\lambda_{ij})y_i^j : \sum_{i\in N\backslash\{j\}}y_i^j = p-1;\ y_i^j\in\{0,1\}, i\in N\backslash\{j\}\right\} \tag{6}$$

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

$$u_1(\Lambda) = \max\left\{\sum_{j\in N}d_j''x_j : \sum_{j\in N}x_j = p;\ x_j\in\{0,1\}, j\in N\right\} \tag{7}$$

We wish to choose the matrix $\Lambda=\{\lambda_{ij}\}$ such that the tightest possible bound $u_1(\Lambda)$ is derived for (5), thus we have the dual problem

$$\begin{aligned} &\text{minimize} && u_1(\Lambda) \\ &\text{subject to} && \lambda_{ij}+\lambda_{ji}=0 \quad i,j\in N. \end{aligned} \tag{8}$$

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 $u_1(\Lambda^k)$. Initially we set $\lambda_{ij}^1=0$ 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 (6) for each value of $j\in N$, and that x_i is the solution vector to (7) for the present value of Λ^k. Then we set

$$\lambda_{ij}^{k+1} := \lambda_{ij}^k + (y_i^jx_i - y_j^ix_j)\frac{p_{ij}+p_{ji}}{2k} \qquad\qquad i,j\in N \tag{9}$$

The motivation for this recursion is, that if $y_i^jx_i=1$ and $y_j^ix_j=0$ then the value of $(d_{ij}+\lambda_{ij}^k)$ contributed to the bound $u_1(\Lambda^k)$ 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 divisior, we achieve smaller adjustments as k increases.

The bound u_2 is obtained by choosing the best value of $u_1(\Lambda)$ thus

$$u_2 = \min_{k=1,\ldots,n}u_1(\Lambda^k) \tag{10}$$

Since each value of $u_1(\Lambda^k)$ 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 (9), the bound 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 u_2 is tighter than the bound u_1 given by (4).*

Proof 1 *Since Λ^1 has $\lambda_{ij} = 0$, we immediately observe that $u_2 \leq u_1(\Lambda^1) = u_1$. The example in Figure 1 shows that situations occur where $u_2 < u_1$.*

In the remaining algorithm, we choose the matrix Λ for which the smallest value of u_2 was obtained, and consider $(d_{ij} + \lambda_{ij})$ as being the new distances in the reformulated problem.

2.1 The main branch-and-bound algorithms

Our branch-and-bound algorithm is based on the upper bounding procedure described in the previous section. At the root node of the branching tree, we apply the reformulation algorithm (9) 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 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 detailes of the branch-and-bound algorithm are outlined in the following sections.

Heuristics

In order to derive a good initial solution, we implemented the heuristic algorithm for QKP devised by Billionnet and Calmels [1], and Caprara, Pisinger, Toth [2]. 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 varible 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 (9) we apply the 2-opt algorithm for the present solution vector x to (7). As will be shown in the computational experiments, this approach gives very good initial solutions.

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 $u_2^{x_i=0}$ be an upper bound on (1) with the additional constraint that $x_i = 0$. If $u_2^{x_i=0} \leq z$ 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 $u_2^{x_i=1}$ be an upper bound on (1) with additional constraint that $x_i = 1$. If $u_2^{x_i=1} \leq z$ 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 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 $d_i i$ should be added to the objective function.

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 u_2 bound (10). 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

$$D_i = \max \left\{ d_{ii} + \sum_{j \in N \backslash \{i\}} (d_{ij} + d_{ji}) y_j^i : \sum_{j \in N \backslash \{i\}} y_j^i = p - 1;\ y_j^i \in \{0,1\}, j \in N \backslash \{i\} \right\} \qquad (11)$$

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 (6) but it turned out that the D_i values are more appropriate as a guideline for the branching process. As seen from Figure 1 (right) the reformulations 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.

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 u_2 according to (6) and (7) 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:

$_i{}^j$	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

Figure 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 buttom 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.

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 varable 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 u_2 is thus also derived in $O(n)$ time.

2.2 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 euclideam 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

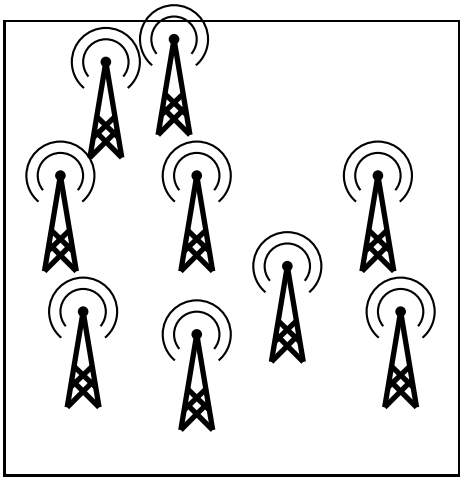

Figure 3: An example of the geometrical problem, where the n possible facilities (in this case: radio transmitters) are randomly placed in a rectangle of size 100×100, and euclidean distances are used for d_{ij}.

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 [6] (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 *assymetric exponential distribution* is also considered in Kincaid [6] (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 Digital workstation 500au with a 500 MHz 21164 CPU (SPECint95 value of 15.7).

First, Table 1 presents the quality of the upper bound u_1 given by (4). The deviation in percent is derived as $100(u_1 - z^*)/z^*$, where z^* is the optimal solution value. A dash in the table indicates that the optimal solution value z^* was not found within the time limit, and thus

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	13.06	11.72	19.47	41.96	11.51	15.15
20	15.29	10.50	23.91	35.60	17.83	23.51
30	16.10	11.09	25.31	34.48	17.77	22.52
40	15.41	7.03	23.56	29.41	11.23	13.43
50	14.75	10.10	24.20	23.88	26.54	34.68
60	28.85	8.68	32.43	36.84	18.77	—
70	18.32	9.90	—	—	—	—
80	12.03	9.30	—	—	—	—
90	—	9.48	—	—	—	—
100	—	17.45	—	—	—	—
150	—	8.01	—	—	—	—
200	—	9.14	—	—	—	—

Table 1: Relative deviation of initial upper bound u_1 in pct. Average of 10 instances.

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	—
70	10.79	3.20	—	—	—	—
80	6.93	2.76	—	—	—	—
90	—	2.92	—	—	—	—
100	—	7.10	—	—	—	—
150	—	2.38	—	—	—	—
200	—	2.78	—	—	—	—

Table 2: Relative deviation of upper bound u_2 in pct. Average of 10 instances.

no relative deviation can be given. It is seen that the upper bound is not very tight since the deviation is about 10–20% for the two geometrical problems and becomes even worse for the other problems.

The tighter upper bound u_2 is considered in Table 2. For the geometrical problems, the deviation drops to about half the value, and becomes as low as 2–7% for the weighted geometrical problems. There is also a reasonable improvement for the other types of instances, in particular for the asymetric exponential problems.

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	—
70	0.0	0.0	—	—	—	—
80	0.0	0.0	—	—	—	—
90	—	0.0	—	—	—	—
100	—	0.0	—	—	—	—
150	—	0.0	—	—	—	—
200	—	0.0	—	—	—	—

Table 3: Relative deviation of initial heuristic solution in pct. Average of 10 instances.

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	13	20
60	0	16	9	6	2	—
70	0	14	—	—	—	—
80	0	20	—	—	—	—
90	—	13	—	—	—	—
100	—	3	—	—	—	—
150	—	27	—	—	—	—
200	—	18	—	—	—	—

Table 4: Number of variables fixed at their optimal values. Average of 10 instances.

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	0.02	0.01	0.02	0.03	0.04	0.04
40	0.11	0.02	0.30	0.83	1.55	4.42
50	1.87	0.07	8.15	1.72	36.90	80.32
60	8.73	0.11	259.92	184.40	13010.40	—
70	117.84	0.20	—	—	—	—
80	485.87	0.43	—	—	—	—
90	—	0.87	—	—	—	—
100	—	1.10	—	—	—	—
150	—	10.93	—	—	—	—
200	—	705.52	—	—	—	—

Table 5: Solution times in seconds as average of 10 instances.

Table 3 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 (7) followed by a 2-opt algorithm), as the optimal solution was found 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, which may be explained by the relatively large gap between the bound u_2 and the optimal solution value.

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	12	7	8	7	9	6
20	140	18	81	227	328	425
30	1654	45	2082	2660	5731	7342
40	11911	26	42842	141149	276825	942041
50	220792	858	1106713	218220	5562368	15368039
60	816402	553	28491542	19646836	1732630354	—
70	9667008	1233	—	—	—	—
80	28885593	7328	—	—	—	—
90	—	16558	—	—	—	—
100	—	13660	—	—	—	—
150	—	123241	—	—	—	—
200	—	7329829	—	—	—	—

Table 6: Number of branch-and-bound nodes. Average of 10 instances.

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 $n = 80$ and weighted geometric problems up to $n = 200$. The remaining problems are somehow more difficult to solve, but still instances up to $n = 50$ are easily solved. For comparison it should be mentioned that Kincaid [6], using metaheuristics, obtained heuristic solution for instances with $n = 25$ and $n = 33$ for the classes EXP and AEXP.

The corresponding number of branching nodes are listed in Table 6. For solving the large instances, several millions of branching nodes are investigated, which shows the importance of deriving tight bounds in $O(n)$ time.

3 The p-dispersion problem

The p-dispersion-sum problem 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 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, $(1 \leq p \leq n)$ 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 $d_{ij} \geq 0$, 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.

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

$$\begin{aligned} \text{maximize} \quad & r \\ \text{subject to} \quad & r x_i x_j \leq d_{ij} \quad i, j \in N \\ & \sum_{j \in N} x_j = p \\ & x_j \in \{0, 1\}, \quad j \in N \\ & r \geq 0 \end{aligned} \tag{12}$$

The first constraint in (12) has the effect that if $x_i x_j = 1$ then $d_{ij} \geq r$. If on the other hand $x_i x_j = 0$ the constraint says $d_{ij} \geq 0$ which is satisfied by assumption.

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

$$r'_j = \max \left\{ r : r y_i^j \leq d_{ij}, i \in N; \ \sum_{i \in N} y_i^j = p; \ y_i^j \in \{0, 1\}, i \in N; \ r \geq 0 \right\} \tag{13}$$

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

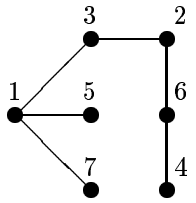

Figure 4: Left: An instance with $n = 7$ facilities and $p = 3$. We find that $r'_1 = 7$, $r'_2 = 9$, $r'_3 = 7$, $r'_4 = 5$, $r'_5 = 5$, $r'_6 = 9$ and $r'_7 = 6$. Hence an upper bound is $u_3 = 7$. Right: For $r = 7$ we get the clique problem represented by the given matrix. Since a clique of size 3 does not exists, $r = 7$ is not a feasible solution value. The optimal solution is obtained with $x_2 = x_4 = x_6 = 1$ having objective value 5.

Then an upper bound u_3 on PDP is found as the solution value to

$$u_3 = \max \left\{ r : rx_j \leq r'_j, j \in N; \ \sum_{j \in N} x_j = p; \ x_j \in \{0,1\}, j \in N; \ r \geq 0 \right\} \tag{14}$$

The problems (13) and (14) 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 u_3 can be derived in $O(n^2)$ time.

Proposition 2 *The value u_3 gives an upper bound on (12).*

Proof 2 *We prove the statement by showing that any feasible solution to (12) is also a feasible solution to (14). Hence assume that the pair (x, r) is a feasible solution to (12). We construct a solution to (13) and (14) for the same value of r. By setting $y^j_i = x_j$ and $r'_j = r$ for $j \in N$ we notice that (y^j, r'_j) is a feasible solution to (13). Hence the pair (x, r) is a solution to (14).*

Since the values r'_j and r in (13) and (14) 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 u_3. 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 u_3.

A Different approach is to decompose the p-dispersion problem (12) into a number of clique problems. Hence for a predefined value of r, define a graph $G = (V, E)$ where V is the set N of locations, and $(i, j) \in E$ if and only if $d_{ij} \geq r$. 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 (12). 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 and use binary search to identify the maximum value of r.

Each clique problem ca be further reformulated to a p-dispersion-sum problem. Let $e_{ij} = 1$ if and only if $(i, j) \in E$. Then we may consider the problem

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in N} \sum_{j \in N} e_{ij} x_i x_j \\ \text{subject to} \quad & \sum_{j \in N} x_j = p \\ & x_j \in \{0, 1\}, \quad j \in N \end{aligned} \tag{15}$$

If a solution of value $z^* = p(p-1)$ is found to (15) we know that a clique of size p exists in the graph (V, E), and thus the current value of r is feasible for (12). Figure 4 illustrates the bound u_3 and the reduction to a clique 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 2.2 with the only exception that d_{ii} always is set to ∞. The generated problems are solved through a transformation of PDSP to PDP as described in Section 3.

n	GEO	WGEO	EXP	AEXP	RAN	DSUB
10	0.02	0.01	0.01	0.01	0.01	0.00
20	0.03	0.03	0.03	0.03	0.03	0.00
30	0.15	0.19	0.11	0.11	0.11	0.01
40	1.63	1.24	0.35	0.51	0.27	0.03
50	16.97	19.67	1.57	0.46	4.48	0.07
60	123.77	116.47	11.29	4.82	48.31	0.13
70	—	—	154.74	90.72	238.60	0.28
80	—	—	—	—	—	0.43
90	—	—	—	—	—	0.69
100	—	—	—	—	—	1.05
150	—	—	—	—	—	9.21
200	—	—	—	—	—	16.79

Table 7: Solution times in seconds for solving the p-dispersion problem. Average values of 10 instances.

Table 7 gives the overall solution time as average of 10 randomly generated instances. 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 seconds 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 factor of about two, which however may be unsufficient in practical applications.

The study of exact algorithms gives an interesting contrast to the results on approximation algorithms. In particular it is intersting to see that several problems can be solved to optimality for n up to 60–80. Moreover, knowing the exact solutions makes it possible to evalute the quality of heuristic solutions. In particular the computational experiments showed that the here presented heuristic is able to find optimal solutions in all the considered instances.

The promising solution times 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. A new algorithm for finding a near-optimal reformulation of the problem in $O(n^3)$ time was presented. This technique may be used for other problems like the quadratic knapsack problem.

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 [5] for solving the clique problems.

References

[1] A. Billionnet and F. Calmels (1996), "Linear Programming for the 0-1 Quadratic Knapsack Problem", *European Journal of Operational Research* **92**, 310–325.

[2] A. Caprara, D. Pisinger, P. Toth (1999), "Exact solution of the quadratic knapsack problem", *INFORMS Journal on Computing*, **11**, 125–137.

[3] E. Erkut (1990), "The discrete p-dispersion problem", *European Journal of Operational Research*, **46**, 48–60.

[4] P. Hansen, I.D. Moon (1988), "Dispersing Facilities on a Network", Presentation at the TIMS/ORSA Joint National Meeting, Washington, D.C.

[5] D.S. Johnson and M.A. Trick (eds.) (1996), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS Press.

[6] R. K. Kincaid (1992), "Good solutions to discrete noxious location problems via metaheuristics", *Annals of Operations Research*, **40**, 265–281.

[7] G. Kortsarz, D. Peleg (1993), On choosing a dense subgraph. In *Proceedings of the 34th Annual IEEE Symposium on Foundation of Computer Science*, 692–701.

[8] J. Krarup, D. Pisinger, F. Plastria (1999), "Discrete location problems with push-pull objectives", Submitted.

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

[10] A. Srivastav, K. Wolf (1998), Finding dense subgraphs with semidefinite programming. In: K. Jansen, J. Rolim (Eds.), *Approximation algorithms for combinatorial optimization*, Lecture Notes in Computer Science, **1444**, 181–191.