Lagrangian Solution of Maximum Dispersion Problems

Şenay Ağca,[1] Burak Eksioglu,[2] and Jay B. Ghosh[3]

[1] *Virginia Polytechnic Institute and State University, Blacksburg, Virginia*

[2] *University of Florida, Gainesville, Florida*

[3] *College of Commerce and Economics, Sultan Qaboos University, P.O. Box 20, Al-Khod, Postal Code 123, Muscat, Oman*

Received July 1997; revised July 1999; accepted September 22, 1999

Abstract: We address the so-called *maximum dispersion* problems where the objective is to maximize the *sum* or the *minimum* of interelement distances amongst a subset chosen from a given set. The problems arise in a variety of contexts including the location of obnoxious facilities, the selection of diverse groups, and the identification of dense subgraphs. They are known to be computationally difficult. In this paper, we propose a Lagrangian approach toward their solution and report the results of an extensive computational experimentation. Our results show that our Lagrangian approach is reasonably fast, that it yields heuristic solutions which provide good lower bounds on the optimum solution values for both the *sum* and the *minimum* problems, and further that it produces decent upper bounds in the case of the *sum* problem. For the *sum* problem, the results also show that the Lagrangian heuristic compares favorably against several existing heuristics. © 2000 John Wiley & Sons, Inc. Naval Research Logistics 47: 97–114, 2000

1. INTRODUCTION

Suppose that we are given a set N of n elements such that the distance between two elements i and j, both in N, is d_{ij} (or d_{ji}); it may be assumed without any loss of generality that $d_{ij}(= d_{ji}) \geq 0$. The so-called *maximum dispersion* problems involve the selection of a subset M of N with m elements, the objective typically being the maximization of some function of the d_{ij} (or d_{ji}) in the selected subset. Two important, and widely addressed, criteria are the *MAXSUM* where one wants to maximize the sum of the distances among the elements of M and the *MAXMIN* where one wants to maximize the minimum of such distances.

The maximum dispersion problems arise in various contexts such as the location of obnoxious facilities, the selection of a maximally diverse group, and the identification of the densest subgraph. In the facility location context, the problems seem to have been introduced by Kuby [15] and followed up, among others, in the work of Erkut [3], Erkut and Neuman [4, 5], Erkut, Baptie, and von Hohenbalken [6], Hansen and Moon [12], Ravi, Rosenkrantz, and Tayi [20], Rahman and Kuby [19], and Hassin, Rubinstein, and Tamir [13]. In the diversity context, the early work appears to be that of Kuo, Glover, and Dhir [16], subsequent work having been done by Ghosh [8] and Glover, Kuo, and Dhir [9, 11]. (Other related works in this context include that of Glover,

Correspondence to: J.B. Ghosh

Kuo, and Dhir [10] on the minimization of group diversity and those of Mingers and O'Brien [17] and Weitz and Lakshminarayanan [23] on the creation of multiple groups with maximal diversity.) In the dense subgraph context, the work of Kortsarz and Peleg [14] and Asahiro et al. [2] are perhaps the most recent examples. The papers by Erkut and Neuman [4, 5], Kuo, Glover, and Dhir [16], Ravi, Rosenkrantz, and Tayi [20] and Hassin, Rubinstein, and Tamir [13] provide good overviews of the different aspects of the research on the problems.

Hansen and Moon [12] and Kuo, Glover, and Dhir [16] independently prove the strong NP-hardness of the MAXSUM problem. Similarly, Erkut [3] and Ghosh [8] prove the strong NP-hardness of the MAXMIN problem. For MAXSUM, Hassin, Rubinstein, and Tamir [13] recently give a polynomial-time approximation algorithm with a performance ratio (which is the smallest possible ratio of an approximate solution value to the optimal) equal to $\frac{1}{2}$. For MAXMIN, Ravi, Rosenkrantz, and Tayi [20] show that it is unlikely that there exists a polynomial-time approximation algorithm with a performance ratio guaranteed to be better than any given constant (less than or equal to 1) when the inter-element distances (d_{ij} or d_{ji}) do not obey the triangle inequality, and a performance ratio better than $\frac{1}{2}$ even when these distances obey the triangle inequality; they actually give an algorithm which achieves the performance ratio of $\frac{1}{2}$ in the latter case. Interesting performance ratios for MAXSUM are also reported by Kortsarz and Peleg [14], Asahiro et al. [2], and Hassin, Rubinstein, and Tamir [13].

Computationally, even though Hansen and Moon [12] indicate that MAXSUM can be solved in a reasonable time for n as large as 50, they do not report any results. From the results that are reported, it appears that this problem can be solved as such for n up to 40 using the specialized branch-and-bound algorithm of Erkut, Baptie, and von Hohenbalken [6] and for n up to 25 using the somewhat more general approach of Ghosh [8]. Erkut [3] and Ghosh [8] report solving MAXMIN for n up to 40 using different algorithms. In addition to the approximation algorithms mentioned above, various heuristic solution approaches to MAXSUM and MAXMIN are proposed by Erkut [3], Erkut and Neuman [5], Erkut, Baptie, and von Hohenbalken [6], Ghosh [8], and Glover, Kuo, and Dhir [11].

In this paper, we turn to a new Lagrangian-based approach for solving the MAXSUM and MAXMIN problems, our focus being primarily on MAXSUM. Our approach produces, in a reasonable amount of CPU time, both a heuristic solution to the problem at hand and an upper bound on its optimal solution value; this enables us to compute a conservative *a posteriori* bound on the heuristic's performance ratio. The reasonable speed and the performance guarantee are what distinguish our approach. Moreover, the upper bound that we generate compares favorably, in terms of CPU time and solution quality, against the common linear programming-based bounds, and can be useful in evaluating heuristic performance.

In the sequel, we present the necessary formulations first. Next, we describe the Lagrangian solution of MAXSUM. We then briefly show how our approach for MAXSUM can be adapted for MAXMIN as well. At this point, we report the results of an extensive computational study, focusing on the performance of the Lagrangian bounds. The results are rather encouraging. In general, we see that the execution times are reasonable. For small-sized problems (where optimal solutions are available), we see that both bounds for MAXSUM and the lower bound for MAXMIN perform quite well. For moderate- to large-sized problems (where optimum solutions are not available), we similarly see that the lower bound for MAXSUM continues to perform reasonably well with respect to the upper bound. For all problem sizes, the lower bound for

MAXSUM outperforms those from several existing heuristics [10]. Finally, we conclude with a summary of our work and some closing thoughts.

2. PROBLEM FORMULATION

Recall that in MAXSUM we want to find a subset M of m elements so that $\sum_{(i,j)\in M} d_{ij}$ is the maximum among all m-element subsets of N. Similarly, in MAXMIN we want to maximize $\min_{(i,j)\in M}\{d_{ij}\}$. In this section, we provide the necessary mathematical formulations for MAXSUM; those for MAXMIN are given later.

Let $x_i = 1$ if i is in M, 0 otherwise $\forall\, i \in N$, and $z_{ij} = 1$ if both i and j are in M, 0 otherwise $\forall\, i, j \in N \ni i < j$. The first formulation we give below is a *0–1 quadratic program* which we later use to generate optimal solutions to MAXSUM.

(P1) Maximize $\sum_{\{(i,j):i,j\in N;i<j\}} d_{ij}x_ix_j$

$$\text{s.t.} \quad \sum_{i\in N} x_i = m, \tag{1}$$

$$x_i \in \{0,1\} \qquad \forall\, i \in N. \tag{2}$$

The next formulation corresponds to a standard *linearization* (refer, for example, to [24]). This linearization has incidentally been recommended by Hansen and Moon [12].

(P2) Maximize $\sum_{\{(i,j):i,j\in N;i<j\}} d_{ij}z_{ij}$

$$\text{s.t.} \quad (1), (2), \text{ and}$$

$$z_{ij} \geq x_i + x_j - 1 \quad \forall\, i, j \in N \ni i < j, \tag{3}$$

$$z_{ij} \leq x_i \quad \forall\, i, j \in N \ni i < j, \tag{4}$$

$$z_{ij} \leq x_j \quad \forall\, i, j \in N \ni i < j, \tag{5}$$

$$z_{ij} \in \{0,1\} \quad \forall\, i, j \in N \ni i < j. \tag{6}$$

Note that constraints (3)–(5) linearize the x_ix_j terms in the objective function of (P1). Specifically, together they ensure that the new variable z_{ij} in (P2) is 1 if and only if x_i and x_j are both 1. Also note that constraints (3) are redundant in our case where $d_{ij} \geq 0$. Similarly, constraints (6) can be replaced by $z_{ij} \geq 0$. We, however, retain this general form for the sake of consistency across formulations.

For the development of the Lagrangian relaxation that we propose in this paper, we resort to a different linearization due to Adams and Sherali [1]; this is seen later to yield a much tighter *continuous relaxation*. The associated formulation, which uses the same notation as (P1) and (P2), is given below.

(P3) Maximize $\sum_{\{(i,j):i,j\in N;i<j\}} d_{ij}z_{ij}$

$$\text{s.t.} \quad (1)\text{–}(6), \text{ and}$$

$$\sum_{\{j:j\in N;j<i\}} z_{ji} + \sum_{\{j:j\in N;j>i\}} z_{ij} = (m-1)x_i \qquad \forall\, i \in N. \tag{7}$$

One can actually try to solve any of the three formulations (P1), (P2), and (P3) directly and get a solution for MAXSUM. However, this approach is feasible only for a small n. For a larger n,

one *per force* turns to a heuristic approach. The Lagrangian relaxation given in the next section is our attempt to develop such an approach.

3. LAGRANGIAN SOLUTION OF MAXSUM

In order to develop a *Lagrangian relaxation* for (P3), we intend to *dualize* constraints (3), (4), and (5). However, the present structure of (7) prevents us from solving the resulting problem easily. That is why we introduce the following redundant constraints:

$$z_{ij} = w_{ij} \quad \forall\, i, j \in N \ni i < j, \tag{8}$$

$$w_{ij} \in \{0, 1\} \quad \forall\, i, j \in N \ni i < j. \tag{9}$$

This allows us to rewrite constraints (7) as follows:

$$\sum_{\{j:j\in N; j<i\}} w_{ji} + \sum_{\{j:j\in N; j>i\}} z_{ij} = (m-1)x_i \quad \forall\, i \in N. \tag{7$'$}$$

(P3) is thus modified to contain constraints (1)–(6), (7′), (8), and (9). Let us call its present form (P3′).

To obtain the Lagrangian relaxation of (P3′), which we call (LR3′), we now dualize constraints (3), (4), (5), and (8) using nonnegative multipliers $\lambda^1_{ij}, \lambda^2_{ij}, \lambda^3_{ij}$ and unrestricted (in sign) multipliers μ_{ij}, respectively. This yields:

(LR3′) Maximize $\sum_{\{(i,j):i,j\in N; i<j\}} [\alpha_{ij} z_{ij} + \beta_{ij} w_{ij}] + \sum_{i\in N} \gamma_i x_i + \delta$

$$\text{s.t.} \quad (1), (2), (6), (7'), \text{ and } (9),$$

where

$$\begin{aligned}
\alpha_{ij} &= d_{ij} + \lambda^1_{ij} - \lambda^2_{ij} - \lambda^3_{ij} + \mu_{ij}, \\
\beta_{ij} &= -\mu_{ij}, \\
\gamma_i &= \sum_{\{j:j\in N; j<i\}} [\lambda^3_{ji} - \lambda^1_{ji}] + \sum_{\{j:j\in N; j>i\}} [\lambda^2_{ij} - \lambda^1_{ij}], \\
\delta &= \sum_{\{(i,j):i,j\in N; i<j\}} \lambda^1_{ij}.
\end{aligned}$$

(LR3′) exhibits a computationally convenient structure. We show below how (LR3′) can be exactly solved in $O(n^2 \log n)$ time. The appendix provides a numerical illustration.

STEP 1. $\forall i \in N$: Sort the elements of the set $\{\alpha_{ij}: j \in N; j > i\} \cup \{\beta_{ji}: j \in N; j < i\}$ in descending order; set the z_{ij} or w_{ij} corresponding to the $(m-1)$ highest values equal to 1 and the rest equal to 0; let θ_i equal the sum of the $(m-1)$ highest values.

STEP 2. Sort the elements of the set $\{\theta_i + \gamma_i: i \in N\}$ in descending order; set the x_i corresponding to the m highest values equal to 1 and the rest equal to 0; reset all z_{ij} or w_{ij} for any i for which x_i is 0.

The time complexity of the above algorithm is easily verified by checking the total effort involved in steps 1 and 2. Its correctness is similarly verified through a simple contradiction argument.

Let LR3′$(\lambda^1, \lambda^2, \lambda^3, \mu)$ be the optimal solution value of (LR3′) for a given set of $\lambda^1_{ij}, \lambda^2_{ij}, \lambda^3_{ij}$, and μ_{ij}. The *Lagrangian dual* for (P3′) is as follows:

(LD3′) Minimize LR3′$(\lambda^1, \lambda^2, \lambda^3, \mu)$

$$\text{s.t.} \quad \lambda^1_{ij}, \lambda^2_{ij}, \lambda^3_{ij} \geq 0 \quad \text{and} \quad \mu_{ij} \quad \text{unrestricted} \qquad \forall i, j \in N \ni i < j.$$

Clearly, the solution value LD3′ of (LD3′) provides an upper bound on the optimal solution value P3′ of (P3′) and, by extension, to those of (P3) and the associated instance of MAXSUM (see, for example, [7]). One may further note that while the continuous relaxation of (LR3′) has a constraint set that is mostly of the *Leontief* type (see, for example, [22]), the presence of the box constraints that substitute constraints (2) in the relaxation destroys the so-called *integrality* property (again see [7]). Therefore, if LP3′ denotes the optimal solution value of the continuous relaxation to (P3′), then we must have: $\mathrm{P3}' \leq \mathrm{LD3}' \leq \mathrm{LP3}'$.

We choose to solve (LD3′) approximately via a *subgradient optimization* algorithm. At each iteration of this algorithm, an instance of (LR3′) is solved and the solution is feasible with respect to (P3′). The latter can thus serve as a heuristic solution to (P3′) and its value H can provide a lower bound on P3′.

Let UB and LB be an upper bound and a lower bound, respectively, on P3′. Also, let a, c, and e be predefined constants. The subgradient optimization algorithm, which is numerically illustrated in the Appendix, can be described as follows:

STEP 0. Set $\lambda^1_{ij}, \lambda^2_{ij}, \lambda^3_{ij}, \mu_{ij} = 0 \quad \forall i, j \in N \ni i < j$; UB $= \infty$ and LB $= 0$.

STEP 1. Solve (LR3′); if LR3′$(\lambda^1, \lambda^2, \lambda^3, \mu) <$ UB, then set UB $=$ LR3′$(\lambda^1, \lambda^2, \lambda^3, \mu)$; if $H >$ LB, then set LB $= H$; if UB has not improved in the last c iterations, then $a \leftarrow a/2$; if $a < e$, then stop, else go to Step 2.

STEP 2. Update the multipliers as follows:

$$s = \sum_{\{(i,j):i,j\in N;i<j\}} [(z_{ij} - x_i - x_j + 1)^2 + (z_{ij} - x_i)^2 + (z_{ij} - x_j)^2 + (z_{ij} - w_{ij})^2],$$

$$t = a(\text{LR3}'(\lambda^1, \lambda^2, \lambda^3, \mu) - \text{LB}).$$

$\forall i, j \in N \ni i < j$:

$$\begin{aligned}
\lambda^1_{ij} &\leftarrow \max\{0, \lambda^1_{ij} - (t/s)(z_{ij} - x_i - x_j + 1)\},\\
\lambda^2_{ij} &\leftarrow \max\{0, \lambda^2_{ij} + (t/s)(z_{ij} - x_i)\},\\
\lambda^3_{ij} &\leftarrow \max\{0, \lambda^3_{ij} + (t/s)(z_{ij} - x_j)\},\\
\mu_{ij} &\leftarrow \mu_{ij} - (t/s)(z_{ij} - w_{ij}).
\end{aligned}$$

Go to Step 1.

Implementation-wise, the parameter a is set to 2; the parameters c and e are set, after an initial tuning, to $\max\{n, 20\}$ and 0.001, respectively. (See [21] for guidelines on the selection of the parameter values.) Finally, the heuristic solution delivered by the subgradient optimization algorithm is subjected to a neighborhood exchange procedure for possible improvement: At any stage of this procedure, let Δ_{ij}, for $x_i \neq x_j$, be the change in the objective function value if x_i

and x_j are complemented (that is, element i is included in or excluded from M, and the opposite is done for element j), and let (i^*, j^*) be a pair such that $\Delta_{i^*j^*} = \max\{\Delta_{ij}: i, j \in N; x_i \neq x_j\}$; if $\Delta_{i^*j^*} > 0$, then set $x_{i^*} = 1 - x_{i^*}$ and $x_{j^*} = 1 - x_{j^*}$; otherwise, if $\Delta_{i^*j^*} \leq 0$, then no further improvement can be made and a local maximum has been found; the exchange mechanism stops.

4. EXTENSION TO MAXMIN

Let y equal $\min_{(i,j)\in M}\{d_{ij}\}$. A mixed 0–1 linear programming formulation for MAXMIN, along the lines of (P3′), is given below:

(P4) Maximize y
s.t. (1)–(6), (7′), (8), and (9),

$$y \leq d_{ij} + (1 - z_{ij})b_{ij} \qquad \forall\, i, j \in N \ni i < j, \tag{10}$$

$$y \geq 0. \tag{11}$$

Note that b_{ij} is a large number—conveniently chosen as $(d^u - d_{ij})$, where d^u is an upper bound on the optimal value of y.

If we dualize (3), (4), (5), and (8) as before and (10), dividing both sides of the inequalities by d^u and using nonnegative multipliers ν_{ij}, we get the following:

(LR4) Maximize $\sum_{\{(i,j):i,j\in N;i<j\}} [\alpha'_{ij}z_{ij}+\beta_{ij}w_{ij}]+\sum_{i\in N}\gamma_i x_i+\delta+\sum_{\{(i,j):i,j\in N;i<j\}} \nu_{ij} + y(1 - \sum_{\{(i,j):i,j\in N;i<j\}} \nu_{ij}/d^u)$

s.t. $(1), (2), (6), (7'), (9),$ and $(11),$

where

$$\alpha'_{ij} = (d_{ij}/d^u - 1)\nu_{ij} + \lambda^1_{ij} - \lambda^2_{ij} - \lambda^3_{ij} + \mu_{ij},$$ and $\beta_{ij}, \gamma_i,$ and δ are as before.

If we further impose the restriction that $\sum_{\{(i,j):i,j\in N;i<j\}} \nu_{ij} = d^u$, the above problem simplifies to:

(LR4′) Maximize $\sum_{\{(i,j):i,j\in N;i<j\}} [\alpha'_{ij}z_{ij} + \beta_{ij}w_{ij}] + \sum_{i\in N}\gamma_i x_i + \delta + d^u$

s.t. $(1), (2), (6), (7'),$ and $(9).$

(LR4′) can be solved in much the same way as (LR3′). The associated Lagrangian dual problem (LD4′) can also be solved similar to (LD3′), except that the new set of multipliers ν_{ij} will have to be accounted for. We leave the details out in the interest of brevity.

Implementationwise, certain points need to be made. Since during the execution of the subgradient optimization algorithm upper bounds on the optimal value of y better than d^u are frequently encountered, it is tempting to replace d^u with such bounds. This changes the original problem definition and marks a departure from tradition. However, we find it advantageous to use this approach. The price we pay is the increase in the run length, which is accommodated by setting $e = 0.0001$. As before, the heuristic solution is improved through neighborhood exchange.

Finally, while we expect that the heuristic solution and the lower bound produced by the subgradient optimization algorithm in this case may be good, we do not expect the upper bound

to be of much value. This is mainly because of the presence of d^u in the formulation of (P4). Thus, in the subsequent section, we focus on the quality of the lower bound only.

5. COMPUTATIONAL RESULTS

We have thus far described the Lagrangian schemes for generating upper and lower bounds for MAXSUM and MAXMIN. We have also noted that the upper bound for MAXMIN is likely to be quite poor. We now report on an extensive computational study that we have conducted to determine the CPU times consumed by our Lagrangian algorithms for MAXSUM and MAXMIN and to test the quality of the upper and lower bounds for MAXSUM and the lower bound for MAXMIN.

We have done three sets of experiments. In the first set, we use small sized problem instances (with n up to 25). For MAXSUM, we compare the tightness of the continuous relaxations of (P2) and (P3) in order to justify our preference for (P3); recall that Hansen and Moon [12] recommend the use of (P2). We also compare the upper and lower bounds for MAXSUM obtained from (LD3$'$) against the optimal solution values. However, for MAXMIN, we report only on the performance of the lower bound generated by the solution of (LD4$'$) and contrast it with the one obtained from the solution of an improved version of the $\frac{1}{2}$-approximate algorithm of Ravi, Rosenkrantz, and Tayi [20]. We note that the latter algorithm is augmented with the same neighborhood exchange mechanism as described earlier and that it is being used essentially as a benchmark.

Next, in the second set, we use moderate to large sized problem instances (with n up to 100). Here, we focus only on MAXSUM, determine the CPU time requirements, and compare the upper and lower bounds obtained from (LD3$'$).

Finally, in the third set, we focus again on MAXSUM and consider all the problem instances from the first two sets. First, we evaluate the performance of two existing heuristics, C2 and D2, given by Glover, Kuo, and Dhir (GKD) in [11] relative to the optimal solution value for small-sized problem instances; we then evaluate the performance of the Lagrangian lower bound with respect to (wrt) C2 and D2 for moderate to large sized instances. Note that C2 and D2 have been among the better performers in the GKD study and that two other heuristics, C1 and D1 in GKD, have data requirements that our test set generation scheme cannot fulfil.

Before describing the test set generation, a few words are in order about the coding of the various algorithms and the computing environment used by us. Optimal solutions for MAXSUM and MAXMIN are generated using FORTRAN codes developed by Ghosh [8] based on the 0–1 quadratic programming solver of Pardalos and Rodgers [18]. The linear programs corresponding to the continuous relaxation of (P2) and (P3) are solved using FORTRAN codes with IMSL routines and C^{++} codes with CPLEX routines (through an easy-to-use interface), respectively. The Lagrangian heuristics as well as the $\frac{1}{2}$-approximate algorithm for MAXMIN are encoded in Pascal; the C2 and D2 heuristics for MAXSUM have been encoded in PASCAL and C^{++}, respectively. All runs, including those for which CPU times are reported, are made on a SPARC Server 1000E operating under Solaris 2.5 at 60 MHz; the exceptions are the C^{++} runs made on Pentium-based PCs running under Windows.

In our first set of experiments, we explore four different problem sizes: $n = 10, 15, 20$, and 25. The reason for choosing these sizes is that their optimal solutions can be obtained in a reasonable amount of CPU time. In the second set of experiments, we opt for three different problem sizes: $n = 50, 75$, and 100; obtaining optimal solutions in these cases is not deemed feasible at this time. For each problem size, 150 instances are generated—25 each for six different distributions of the d_{ij}, covering a wide range of possibilities. The distributions used by us are shown in Figure 1. Note that a problem instance can be completely specified by n, $\{d_{ij}: i, j \in N; i < j\}$ and m.

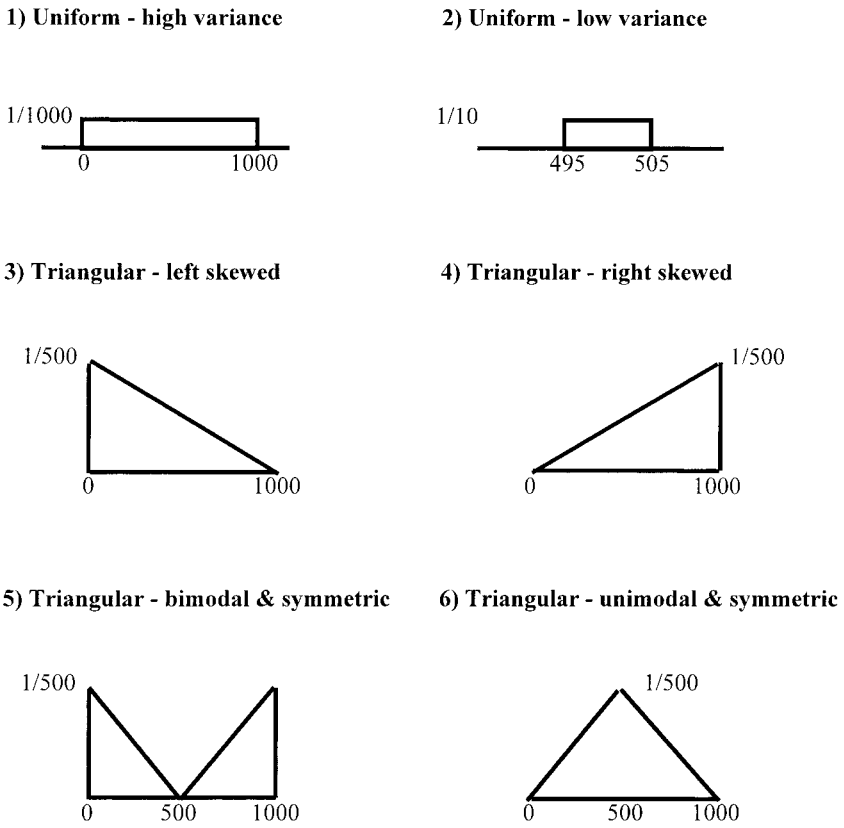

Figure 1. Distance distributions used in test set generation.

Consistent with Erkut [3] and Ghosh [8], we try two different levels of m: $m = 0.2n$ and $0.4n$. However, note that the combination $n = 10$ and $m = 0.2n$ leads to the trivial case of $m = 2$; we do not consider it in our experimentation. One final point needs to be made: Unlike some of the earlier studies including GKD [11], the distances sampled by us are not required to satisfy the triangle inequality.

We report the results of our first set of experiments (for small sized problem instances) in Tables 1–3. Table 1 records our comparison of the tightness of the continuous relaxations of (P2) and (P3) for MAXSUM. This is done by displaying the ratios of the optimal solution values of the relaxations to that of the original 0–1 quadratic programming problem. Clearly, (P3) is seen to yield a much tighter relaxation. We notice that this particular relaxation becomes weaker with increasing n and m and that it performs relatively poorly for distribution types 3 (left skewed), 5 (bimodal and symmetric), and 1 (uniform with high variance). Curiously, the (P2)-based relaxation performs the worst for distribution type 2 (uniform with low variance) where the (P3)-based one performs the best.

Table 2 shows the performance of the Lagrangian upper and lower bounds for MAXSUM derived from (LD3′) and the CPU times in seconds taken by the Lagrangian algorithm. Once again, the ratios of the bounds to the optimal solution value are reported, this time along with the percentage of times the lower bound equals the optimal. In theory, as noted earlier, our upper bound should be at most as much as LP3′ which is shown in Table 1. However, in reality, we see that our upper bound, while mostly close or even equal to, occasionally lags behind the

Table 1. Comparison of the continuous relaxations of (P2) and (P3).

			Performance ratio			
			(P2) Relaxation		(P3) Relaxation	
n	m	Distribution type	Average	Worst	Average	Worst
10	4	1	1.9423	2.1755	1.0331	1.1219
		2	2.9826	2.9875	1.0005	1.0017
		3	1.7451	2.0282	1.0486	1.1373
		4	2.2750	2.4604	1.0233	1.0907
		5	1.7959	2.1085	1.0465	1.1350
		6	2.1454	2.2978	1.0234	1.0985
15	3	1	2.8872	4.1412	1.0291	1.0725
		2	6.9411	6.9516	1.0005	1.0017
		3	3.1095	3.7176	1.0515	1.1822
		4	4.8536	5.2614	1.0075	1.0352
		5	3.6169	4.0092	1.0093	1.0281
		6	4.2333	4.6451	1.0234	1.0778
	6	1	1.9559	2.1100	1.1062	1.1712
		2	2.7873	2.7903	1.0015	1.0026
		3	1.7658	1.9074	1.1147	1.2354
		4	2.2170	2.3285	1.0602	1.1014
		5	1.7979	1.9472	1.1272	1.1922
		6	2.1217	2.3022	1.0731	1.1305
20	4	1	3.6247	3.8602	1.0467	1.1046
		2	6.2848	6.2915	1.0008	1.0018
		3	3.0418	3.4576	1.0743	1.1588
		4	4.5300	4.7609	1.0323	1.0586
		5	3.3920	3.6210	1.0358	1.0736
		6	4.0418	4.2737	1.0512	1.0882
	8	1	1.9482	2.0293	1.1396	1.2016
		2	2.7035	2.7062	1.0020	1.0028
		3	1.8034	1.9107	1.1601	1.2257
		4	2.2041	2.2735	1.0881	1.1421
		5	1.8270	1.9467	1.1812	1.2627
		6	2.1171	2.2483	1.0970	1.1424
25	5	1	3.6029	3.8938	1.0899	1.1475
		2	5.9591	5.9642	1.0015	1.0025
		3	3.1167	3.3520	1.1381	1.2108
		4	4.3788	4.5057	1.0448	1.0658
		5	3.3033	3.5934	1.0548	1.0856
		6	3.9844	4.2771	1.0731	1.1195
	10	1	1.9715	2.1351	1.1705	1.2430
		2	2.6575	2.6583	1.0025	1.0030
		3	1.8427	1.9430	1.2016	1.2552
		4	2.2233	2.2622	1.1096	1.1433
		5	1.8596	1.9520	1.2205	1.3208
		6	2.1433	2.2378	1.1287	1.1814

Table 2. Performance of the upper and lower bounds from (LD3′).

			Performance ratio					
			Upper bound		Lower bound			Average
n	m	Distribution type	Average	Worst	Average	Worst	% Optimal	CPU time (s)
10	4	1	1.0339	1.1229	0.9938	0.9276	84	0.13
		2	1.0004	1.0017	0.9998	0.9980	72	0.11
		3	1.0493	1.1383	0.9988	0.9862	84	0.13
		4	1.0238	1.0907	0.9985	0.9806	84	0.15
		5	1.0466	1.1350	0.9981	0.9663	92	0.14
		6	1.0241	1.0986	1.0000	1.0000	100	0.14
15	3	1	1.0294	1.0725	0.9920	0.9145	76	0.39
		2	1.0003	1.0013	0.9999	0.9987	92	0.52
		3	1.0522	1.1822	0.9866	0.9193	68	0.41
		4	1.0076	1.0358	0.9972	0.9778	76	0.37
		5	1.0096	1.0279	0.9981	0.9682	84	0.41
		6	1.0238	1.0778	0.9933	0.8881	84	0.34
	6	1	1.1067	1.1716	0.9909	0.9453	64	0.48
		2	1.0015	1.0027	0.9999	0.9992	68	0.46
		3	1.1153	1.2356	0.9950	0.9620	72	0.48
		4	1.0604	1.1016	0.9963	0.9749	68	0.51
		5	1.1277	1.1926	0.9967	0.9631	84	0.51
		6	1.0735	1.1306	0.9962	0.9675	76	0.49
20	4	1	1.0477	1.1063	0.9873	0.8960	60	1.02
		2	1.0007	1.0017	0.9996	0.9980	52	1.12
		3	1.0753	1.1602	0.9842	0.8887	64	0.93
		4	1.0327	1.0586	0.9908	0.9542	48	1.06
		5	1.0361	1.0740	0.9889	0.9226	60	1.01
		6	1.0526	1.0885	0.9920	0.9503	72	0.95
	8	1	1.1404	1.2018	0.9917	0.9565	64	0.94
		2	1.0020	1.0028	0.9999	0.9993	72	0.95
		3	1.1609	1.2263	0.9912	0.9094	56	0.95
		4	1.0884	1.1424	0.9959	0.9578	72	0.94
		5	1.1818	1.2633	0.9908	0.9550	64	1.00
		6	1.0974	1.1427	0.9938	0.9615	56	0.98
25	5	1	1.0905	1.1476	0.9914	0.9497	60	2.09
		2	1.0015	1.0024	0.9996	0.9988	40	2.26
		3	1.1389	1.2122	0.9835	0.9204	56	2.17
		4	1.0451	1.0660	0.9915	0.9569	40	2.23
		5	1.0550	1.0857	0.9937	0.9520	72	2.23
		6	1.0740	1.1210	0.9847	0.9280	48	2.22
	10	1	1.1710	1.2435	0.9928	0.9660	56	1.94
		2	1.0025	1.0030	0.9999	0.9995	76	1.95
		3	1.2021	1.2554	0.9976	0.9839	72	2.03
		4	1.1099	1.1438	0.9918	0.9637	36	1.99
		5	1.2210	1.3214	0.9867	0.9434	48	2.09
		6	1.1291	1.1816	0.9907	0.9531	48	2.01

Table 3. Performance of the lower bound from (LD4′) against the benchmark heuristic.

			Performance ratio					
			Benchmark		Lagrangian			Lagrangian Average
n	m	Distribution type	Average	Worst	Average	Worst	% Optimal	CPU time (s)
10	4	1	0.8977	0.5633	0.9956	0.8894	96	1.77
		2	0.9985	0.9940	0.9998	0.9980	92	0.60
		3	0.9327	0.5783	0.9992	0.9809	96	1.36
		4	0.9606	0.8390	0.9960	0.9428	88	1.83
		5	0.7809	0.3329	0.9927	0.8184	96	1.58
		6	0.9076	0.6629	0.9901	0.7586	92	1.70
15	3	1	0.8885	0.6701	0.9900	0.9266	80	5.80
		2	0.9984	0.9960	0.9999	0.9980	96	1.23
		3	0.8986	0.5959	0.9910	0.8658	88	7.68
		4	0.9612	0.8322	0.9958	0.9711	80	5.70
		5	0.9235	0.4340	0.9962	0.9388	80	6.87
		6	0.9310	0.7358	0.9948	0.9239	92	8.40
	6	1	0.8149	0.5069	0.9601	0.7079	80	0.70
		2	0.9984	0.9940	0.9991	0.9960	64	3.77
		3	0.8264	0.3611	0.9760	0.6991	84	8.74
		4	0.8863	0.6556	0.9886	0.9079	76	10.25
		5	0.7541	0.4130	0.9707	0.6215	76	20.66
		6	0.9278	0.7689	0.9827	0.8401	72	9.45
20	4	1	0.8537	0.6083	0.9503	0.7419	44	14.11
		2	0.9979	0.9940	0.9979	0.9940	40	3.58
		3	0.8767	0.6449	0.9413	0.6781	56	22.56
		4	0.9308	0.8231	0.9498	0.8416	32	5.54
		5	0.8853	0.4246	0.9658	0.8982	44	12.22
		6	0.9168	0.7516	0.9829	0.9013	72	12.19
	8	1	0.8373	0.4714	0.9118	0.6459	44	17.59
		2	0.9985	0.9960	0.9981	0.9960	20	7.08
		3	0.8684	0.4913	0.9047	0.5541	48	17.79
		4	0.9161	0.7769	0.9552	0.7830	40	18.45
		5	0.8262	0.3410	0.9232	0.5940	52	18.82
		6	0.8972	0.7228	0.9555	0.8561	40	17.64
25	5	1	0.8625	0.6288	0.9155	0.7841	28	28.10
		2	0.9977	0.9920	0.9970	0.9940	12	2.76
		3	0.8210	0.5289	0.8741	0.6361	28	32.73
		4	0.9006	0.7133	0.9202	0.7622	8	1.25
		5	0.7986	0.3233	0.9492	0.8473	36	29.66
		6	0.9262	0.7645	0.9319	0.7636	24	13.86
	10	1	0.7872	0.4767	0.8442	0.5400	16	32.02
		2	0.9986	0.9960	0.9975	0.9960	0	5.19
		3	0.7749	0.4295	0.8546	0.6783	16	34.16
		4	0.8697	0.5712	0.9251	0.6667	36	25.31
		5	0.7989	0.4333	0.8721	0.5701	32	33.49
		6	0.8704	0.6227	0.9364	0.7672	28	32.49

linear programming bound. The average percentage deviation from the optimal is 22.1% in the worst case ($n = 25$, $m = 10$, and distribution type 5); the worst deviation is 32.14% (for the same parameter settings). The deviation seems to grow worse as n and m increase. Generally, distribution types 3, 5, and 1 are where the bound performs relatively poorly.

As for the lower bound, we see that it has performed extremely well. The average percentage deviation in this case is 1.65% in the worst case; the worst deviation is 11.19%. One natural question is to what extent this is attributable to the exchange mechanism. The answer seems to be considerable, since without the exchange the worst average deviation is 12.6%. However, we should note that this can drop to 2.18% when the parameter LB in the subgradient optimization algorithm is initialized with a heuristic solution value (obtained in this instance from the $\frac{1}{2}$-approximate algorithm of Hassin, Rubinstein, and Tamir [13]) and no exchange is used. As n increases, the deviation from the optimal increases; however, as m increases, it appears that the deviation does not change significantly (in fact, it seems to decrease slightly). Here also distribution types 3, 5, and 1 lead to relatively poor results.

Table 3 presents our results for the Lagrangian solution of MAXMIN. Since the upper bound in this case is unlikely to be of any practical value (it can be as high as 6.23 times the optimal), we do not report it. Instead, we report the performance of the lower bound and compare it with the benchmark heuristic mentioned earlier. As before, the ratios of the bounds to the optimal are given, as are the percentage of times the Lagrangian bound equals the optimal and the CPU times needed to obtain it. The results, of course, are not as impressive as in the MAXSUM case, but they are significantly better than those for the benchmark. (To be fair, however, we should admit that the benchmark is orders of magnitude faster than the Lagrangian.) The average percentage deviation is 15.58% in the worst case; the worst deviation is 46%. To correctly interpret the quality of the bound, one should also look at the percentage of times the bound equals the optimal. We have seen that the relatively large average deviations are often due to a few outliers; thus, the median would perhaps have been a better measure of quality here. As for the impact of the exchange mechanism, we note that the worst average percentage deviation is 33.71% without it. (We may mention that our neighborhood structure does not work too well with MAXMIN [8] and that it has performed especially poorly for the benchmark.) We have not tested if this figure could have been improved through proper initialization (partly because the heuristic intended for that purpose is also the benchmark). The performance of the bound deteriorates as n and m increase. The unfavorable distribution types are 1 and 3 this time.

Brief mention should be made about the CPU times in Table 3. Not only are these times much higher on the average than those in the case of MAXSUM, they also exhibit a relatively high variability. To some extent this can be attributed to the algorithm design in general and the stopping criteria (the low value of the parameter e used) in particular.

The results of our second set of experiments (for moderate to large sized problem instances) on MAXSUM are reported in Table 4. In absence of optimal solution values, we compare the performance of the Lagrangian lower and upper bounds against each other. The ratio of the two bounds is shown along with the CPU time. We see that the CPU times are quite reasonable. We also see that the worst average percentage deviation of the lower bound from the upper bound is 46.92%, whereas the worst deviation is 48.98%. The behavior of the performance bound with increasing n and m is similar to that observed in Table 2. Also, as in Table 2, distribution types 3, 5, and 1 are where the bound performs poorly. These distribution types give us poor lower and upper bounds (see Table 2), and, therefore, it is not at all surprising that they give us poor lower to upper bound ratios as well (see Table 4).

Note that the deviations in Table 4, while acceptable as performance guarantees for our Lagrangian heuristic, are rather conservative and can grossly underestimate its true performance.

Table 4. Lower to upper bound ratio for (LD3′).

n	m	Distribution type	Lower to upper bound ratio		Average CPU time (s)
			Average	Worst	
50	10	1	0.7866	0.7409	2.88
		2	0.9961	0.9953	2.94
		3	0.7328	0.6824	2.86
		4	0.8697	0.8171	2.96
		5	0.8018	0.7633	2.94
		6	0.8301	0.7748	2.92
	20	1	0.7593	0.7249	3.03
		2	0.9959	0.9947	2.87
		3	0.7373	0.7057	2.98
		4	0.8324	0.8133	2.89
		5	0.7206	0.6979	3.07
		6	0.8144	0.7927	2.98
75	15	1	0.7410	0.7131	11.96
		2	0.9952	0.9947	12.87
		3	0.6954	0.6500	11.75
		4	0.8387	0.8101	12.83
		5	0.7424	0.7060	13.06
		6	0.7871	0.7670	12.40
	30	1	0.7243	0.7102	17.90
		2	0.9799	0.9794	13.47
		3	0.7063	0.6907	17.14
		4	0.8089	0.7954	13.93
		5	0.6840	0.6546	16.62
		6	0.7841	0.7578	15.72
100	20	1	0.6934	0.6274	19.54
		2	0.9931	0.9923	19.72
		3	0.6330	0.5102	19.64
		4	0.8067	0.7730	19.52
		5	0.7043	0.6689	19.62
		6	0.7170	0.6469	19.64
	40	1	0.6181	0.5990	17.60
		2	0.9882	0.9858	19.84
		3	0.5308	0.5200	18.22
		4	0.7430	0.7266	17.65
		5	0.6050	0.5972	17.48
		6	0.6691	0.6560	18.10

To estimate what the performance ratio is truly like, we perform an extrapolation for the case where $n = 50, m = 10$, and the distribution type is 1. We regress the ratio of the Lagrangian upper bound to the optimal (call it $\rho_{\mathrm{UB/OPT}}$) on square roots of n and m (using the data for distribution type 1 from Table 2) and obtain the following relationship (with an adjusted $R^2 = 0.9668$):

$$\rho_{\mathrm{UB/OPT}} = 0.8112 + 0.0112 \text{ sqrt}(n) + 0.0980 \text{ sqrt}(m).$$

Table 5. Performance of GKD heuristics C2 and D2.

			Performance ratio					
			C2			D2		
n	m	Distribution type	Average	Worst	% Optimal	Average	Worst	% Optimal
10	4	1	0.9772	0.9016	48	0.9483	0.8705	24
		2	0.9997	0.9980	64	0.9985	0.9957	24
		3	0.9769	0.8336	60	0.9307	0.7734	24
		4	0.9775	0.8759	56	0.9517	0.8252	20
		5	0.9442	0.7610	44	0.9641	0.8173	44
		6	0.9708	0.7833	56	0.9566	0.8456	40
15	3	1	0.9604	0.8698	32	0.9106	0.7446	20
		2	0.9992	0.9967	40	0.9964	0.9927	0
		3	0.9488	0.8463	40	0.8690	0.6720	16
		4	0.9652	0.9103	16	0.9022	0.7442	4
		5	0.9615	0.8616	8	0.9231	0.8099	4
		6	0.9697	0.8802	40	0.8982	0.7360	24
	6	1	0.9676	0.8603	36	0.9668	0.8998	28
		2	0.9995	0.9983	24	0.9990	0.9966	8
		3	0.9591	0.8743	28	0.9657	0.8733	36
		4	0.9767	0.9256	20	0.9758	0.9166	24
		5	0.9591	0.8950	20	0.9621	0.8772	32
		6	0.9750	0.9033	32	0.9672	0.9287	4
20	4	1	0.9305	0.8444	16	0.9400	0.8408	12
		2	0.9990	0.9967	36	0.9982	0.9957	8
		3	0.9324	0.7625	24	0.9050	0.7212	20
		4	0.9796	0.9510	16	0.9440	0.8593	0
		5	0.9624	0.8646	28	0.9408	0.7512	12
		6	0.9690	0.8620	32	0.9395	0.8550	12
	8	1	0.9712	0.9066	32	0.9801	0.9080	28
		2	0.9994	0.9982	24	0.9996	0.9984	28
		3	0.9680	0.8752	20	0.9759	0.9094	20
		4	0.9817	0.9220	28	0.9823	0.9509	16
		5	0.9701	0.8932	24	0.9794	0.9180	32
		6	0.9780	0.9007	24	0.9824	0.9548	24
25	5	1	0.9549	0.8322	20	0.9490	0.8536	16
		2	0.9994	0.9980	32	0.9983	0.9968	8
		3	0.9617	0.8199	28	0.9213	0.7043	8
		4	0.9803	0.9219	36	0.9579	0.9037	8
		5	0.9602	0.8673	28	0.9502	0.8599	12
		6	0.9705	0.8921	24	0.9477	0.8842	20
	10	1	0.9700	0.8869	28	0.9849	0.9404	20
		2	0.9996	0.9989	16	0.9996	0.9990	4
		3	0.9759	0.9140	24	0.9767	0.9155	16
		4	0.9831	0.9465	4	0.9860	0.9550	8
		5	0.9683	0.9062	20	0.9761	0.9418	4
		6	0.9867	0.9614	20	0.9833	0.9517	12

Table 6. Relative performance of the lower bound from (LD3′) wrt GKD heuristics C2 and D2.

			Relative performance ratio							
			wrt C2				wrt D2			
n	m	Distribution type	Average	Worst	Best	% Better	Average	Worst	Best	% Better
50	10	1	1.0184	0.9529	1.0812	76	1.0232	0.9717	1.0843	76
		2	1.0004	0.9995	1.0016	76	1.0001	0.9987	1.0009	52
		3	1.0239	0.9120	1.1954	80	1.0310	0.9402	1.1165	68
		4	1.0060	0.9467	1.0680	56	1.0123	0.9713	1.0623	60
		5	1.0022	0.9317	1.0858	44	1.0166	0.9391	1.0869	64
		6	1.0144	0.9692	1.0549	72	1.0107	0.9719	1.0617	56
	20	1	1.0287	0.9826	1.0858	92	1.0370	1.0163	1.0714	100
		2	1.0011	1.0005	1.0016	100	1.0011	1.0006	1.0018	100
		3	1.0300	1.0003	1.0738	100	1.0342	0.9900	1.0783	96
		4	1.0286	1.0022	1.0574	100	1.0330	1.0017	1.0755	100
		5	1.0160	0.9756	1.0644	72	1.0301	0.9964	1.0699	92
		6	1.0321	1.0001	1.0681	100	1.0333	1.0176	1.0733	100
75	15	1	1.0255	0.9728	1.0673	88	1.0327	0.9874	1.0775	92
		2	1.0005	0.9997	1.0013	88	1.0004	0.9993	1.0012	68
		3	1.0114	0.9544	1.0740	68	1.0253	0.9476	1.1214	80
		4	1.0111	0.9818	1.0425	80	1.0156	0.9848	1.0518	76
		5	1.0252	0.9728	1.1227	80	1.0251	0.9620	1.1023	80
		6	1.0162	0.9824	1.0652	80	1.0219	0.9856	1.0639	92
	30	1	1.0362	1.0107	1.0702	100	1.0452	1.0168	1.0698	100
		2	1.0165	1.0161	1.0171	100	1.0165	1.0162	1.0170	100
		3	1.0287	0.9927	1.0566	92	1.0376	1.0075	1.0711	100
		4	1.0306	1.0136	1.0587	100	1.0360	1.0234	1.0741	100
		5	1.0354	1.0068	1.0809	100	1.0415	1.0163	1.0777	100
		6	1.0366	1.0124	1.0740	100	1.0444	1.0095	1.0756	100
100	20	1	1.0172	0.9838	1.0836	72	1.0274	0.9672	1.0722	88
		2	1.0004	0.9998	1.0009	92	1.0005	0.9999	1.0010	92
		3	1.0287	0.9788	1.0730	88	1.0299	0.9749	1.0930	80
		4	1.0155	0.9832	1.0440	88	1.0210	0.9936	1.0438	92
		5	1.0216	0.9910	1.0902	84	1.0290	0.9817	1.0818	84
		6	1.0192	0.9900	1.0592	80	1.0260	0.9951	1.0552	96
	40	1	1.0264	0.9998	1.0582	96	1.0347	1.0107	1.0529	100
		2	1.0058	1.0043	1.0086	100	1.0059	1.0045	1.0086	100
		3	1.0324	1.0070	1.0645	100	1.0399	1.0148	1.0764	100
		4	1.0328	1.0120	1.0605	100	1.0374	1.0190	1.0563	100
		5	1.0308	0.9939	1.0547	96	1.0425	1.0186	1.0667	100
		6	1.0323	1.0166	1.0536	100	1.0406	1.0230	1.0647	100

For $n = 50, m = 10$ and distribution type 1, this yields $\rho_{\mathrm{UB/OPT}} = 1.2003$. The reported ratio $(\rho_{\mathrm{LB/UB}})$ in Table 4 for this situation is 0.7866. Multiplying $\rho_{\mathrm{UB/OPT}}$ by $\rho_{\mathrm{LB/UB}}$, we get an estimate of the true (but unknown) performance ratio, $\rho_{\mathrm{LB/OPT}}$, as 0.9442; this represents only a 5.58% deviation (as opposed to the 21.34% reported in Table 4).

Tables 5 and 6 record the results of our third set of experiments on MAXSUM using heuristics C2 and D2 of GKD for problem instances of all sizes. We have no interest in CPU times here

since C2 and D2 are polynomial-time heuristics and can generally be assumed to be faster than our Lagrangian. (However, note that the Lagrangian is also polynomial-time for a fixed number iterations and that it can be reasonably fast as demonstrated.)

Table 5 reports, for small instances, the worst and average case performance ratios for C2 and D2 measured with respect to the optimal solution value; also reported are the percentage of times these heuristics find the optimal solution. The average percentage deviation for C2 is 6.95% in the worst case while the absolutely worst deviation is 23.90%; the percentage of times an optimal solution has been found is in the range 4–64%. The corresponding numbers for D2 are 13.10%, 32.8%, and 0–44%, respectively. These figures mildly contradict what have been found by GKD [11]. Anyhow, the Lagrangian lower bound exhibits (see Table 2) much better performance; its numbers are 1.65%, 11.19%, and 36–100%, respectively. This shows that the Lagrangian is generally much superior to C2 and D2.

Finally, Table 6 reports, for moderate to large instances, the worst, average, and best relative performance ratios for the Lagrangian lower bound measured with respect to (wrt) C2 and D2 (note that a value of 1 or higher in this case indicates superior performance of the Lagrangian); the percentage of times the Lagrangian yields a *strictly* better solution is also reported. On the average, the Lagrangian is always better than C2; the average percentage deviation is in the range $+0.04\%$–$+3.62\%$. The best deviation is $+19.54\%$ and the worst is -8.80%; the percentage of times the Lagrangian is *strictly* better than C2 ranges from 44% to 100%. Again, the Lagrangian is always better than D2 on the average; the average percentage deviation in this case is in the range $+0.01\%$–4.52%. The best deviation is $+12.14\%$ and the worst is -6.09%; the Lagrangian is *strictly* better than D2 52–100% of the times. This establishes the general superiority of the Lagrangian over both C2 and D2. However, the distinction between C2 and D2 is not very clear here.

6. CONCLUSION

We have addressed two versions of the maximum dispersion problem and proposed Lagrangian solutions for them. Even for moderate- to large-sized problem instances, our heuristics execute in a reasonable amount of CPU time and are thus quite suitable for practical use. In terms of quality, for MAXSUM, the upper and lower bounds obtained by us are quite good, whereas, for MAXMIN, the lower bound alone is of interest. In both cases, the heuristic solutions corresponding to the lower bounds are competitive with the existing heuristics (based on the results reported on their performance and the comparisons performed by us). In addition, the upper bound for MAXSUM provides a conservative *a posteriori* guarantee for the Lagrangian heuristic and can also be useful in the evaluation of the solution quality of any heuristic for moderate to large sized problem instances.

A number of issues remain open for further exploration. On the theoretical side, unlike MAXMIN, there are no results for MAXSUM with respect to the existence of a polynomial-time relative approximation algorithm for the situation where the triangle inequality does not hold. Likewise, on the computational side, there is a need to study the performance of generic heuristic strategies for both MAXSUM and MAXMIN in a more comprehensive way.

APPENDIX

We provide here numerical illustrations of the algorithms that we use to solve (LR3′) and (LD3′), respectively. For this purpose, we consider an instance of the MAXSUM problem with $n = 5, m = 3$, and $\{d_{ij}\colon i, j \in N; i < j\} = \{4, 2, 3, 1, 2, 3, 5, 1, 4, 3\}$.

Let us first look at the solution of (LR3′). Assume that $\lambda_{ij}^1, \lambda_{ij}^2, \lambda_{ij}^3$, and μ_{ij} are equal to 0 for all i and j, as they are in the first iteration of the subgradient optimization algorithm for solving (LD3′). It is easily seen in this case that $\alpha_{ij} = d_{ij}$ and $\beta_{ij} = 0$ for all i and j, $\gamma_i = 0$ for all i, and $\delta = 0$. Thus, for $i = 1$, the coefficients of z_{12}, z_{13}, z_{14}, and z_{15} are $4, 2, 3$, and 1, respectively. We accordingly set $z_{12} = z_{14} = 1$ and $z_{13} = z_{15} = 0$ and get $\theta_1 = 7$. Proceeding similarly, we get $z_{24} = z_{25} = 1$ and $w_{12} = z_{23} = 0$ for $i = 2$ with $\theta_2 = 8$; $z_{34} = z_{35} = 1$ and $w_{13} = w_{23} = 0$ for $i = 3$ with $\theta_3 = 5$; $z_{45} = w_{14} = 1$ and $w_{24} = w_{34} = 0$ for $i = 4$ with $\theta_4 = 3$; and $w_{15} = w_{25} = 1$ and $w_{35} = w_{45} = 0$ for $i = 5$ with $\theta_5 = 0$. At this stage, based on the above θ_i values, we set $x_1 = x_2 = x_3 = 1$ and $x_4 = x_5 = 0$. We also set $z_{12} = z_{14} = z_{24} = z_{25} = z_{34} = z_{35} = 1$; the remaining z_{ij} and w_{ij} variables are set to 0. What we have just obtained constitutes an optimal solution to our instance of (LR3′). Its objective function value is 20 and that is an upper bound on the optimal solution value of problems (P1)–(P3). It also gives us the feasible subset $\{1, 2, 3\}$ as a heuristic solution to problems (P1)–(P3); the total distance of this solution, which is 8, is a lower bound on the optimal.

Let us now turn to the solution of (LD3′) using the subgradient optimization algorithm. We consider only the first iteration. As before, the multipliers are all set to 0, and UB $= \infty$ and LB $= 0$. We now solve (LR3′) and get the solution given above. Thus, UB is set to 20 and LB to 8. At this stage, we compute s, based on the values of x_i, z_{ij}, and w_{ij} from above and, get $s = 24$. Using $a = 2$, we similarly get $t = 24$. The multipliers are then updated as follows: $\lambda_{13}^1 = \lambda_{23}^1 = \lambda_{14}^3 = \lambda_{24}^3 = \lambda_{25}^3 = \lambda_{34}^3 = \lambda_{35}^3 = 1$; $\mu_{12} = \mu_{14} = \mu_{24} = \mu_{25} = \mu_{34} = \mu_{35} = -1$; and, all other multipliers remain at 0. This done, we go back to resolve (LR3′). The process continues until the stopping criterion is satisfied.

ACKNOWLEDGMENTS

Thanks are due to Panos Pardalos and Greg Rodgers for letting us use their 0–1 quadratic programming code. Thanks are also due to Ossama Kettani for his help with the CPLEX solution of the linear programming relaxations through EZMOD (an easy-to-use interface developed by him). Finally, our revisions of the paper leading to the present version have benefited significantly from the helpful comments of an associate editor and two anonymous referees.

REFERENCES

[1] W.P. Adams and H.D. Sherali, A tight linearization and an algorithm for zero-one quadratic programming problems, Manage Sci 32 (1986), 1274–1290.

[2] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama, Greedily finding a dense subgraph, Working Paper, Department of Computer Science, Kyushu University, Japan, 1996.

[3] E. Erkut, The discrete p-dispersion problem, Eur J Oper Res 46 (1990), 48–60.

[4] E. Erkut and S. Neuman, Analytical models for locating undesirable facilities, Eur J Oper Res 40 (1989), 275–291.

[5] E. Erkut and S. Neuman, Comparison of four models for dispersing facilities, INFOR 29 (1991), 68–85.

[6] E. Erkut, T. Baptie, and B. von Hohenbalken, The discrete p-maxian location problem, Comput Oper Res 17 (1990), 51–61.

[7] M.L. Fisher, The Lagrangian relaxation method for solving integer programming problems, Manage Sci 27 (1981), 1–18.

[8] J.B. Ghosh, Computational aspects of the maximum diversity problem, Oper Res Lett 19 (1996), 175–181.

[9] F. Glover, C.-C. Kuo, and K.S. Dhir, A discrete optimization model for preserving biological diversity, Appl Math Model 19 (1995), 696–701.

[10] F. Glover, C.-C. Kuo, and K.S. Dhir, Integer programming and heuristic approaches to the minimum diversity problem, J Bus Manage 4 (1997), 93–111.

[11] F. Glover, C.-C. Kuo, and K.S. Dhir, Heuristic algorithms for the maximum diversity problem, J Inf Optim Sci 19 (1998), 109–132.

[12] P. Hansen and I.J. Moon, Dispersing facilities on a network, Cah CERO, 36 (1994), 221–234.

[13] R. Hassin, S. Rubinstein, and A. Tamir, Approximation algorithms for maximum facility dispersion, Working Paper, Department of Statistics and Operations Research, School of Mathematical Sciences, Tel-Aviv University, Israel, 1996.

[14] G. Kortsarz and D. Peleg, On choosing a dense subgraph, Proc 34th IEEE Annu Symp Found Comput Sci, Palo Alto, 1993, pp. 692–701.
[15] M.J. Kuby, Programming models for facility dispersion: The p-dispersion and maximum dispersion problems, Geogr Anal 19 (1987), 315–329.
[16] C.-C. Kuo, F. Glover, and K.S. Dhir, Analyzing and modeling the maximum diversity problem by zero–one programming, Decision Sci 24 (1993), 1171–1185.
[17] J. Mingers and F.A. O'Brien, Creating student groups with similar characteristics: A heuristic approach, OMEGA 23 (1995), 313–332.
[18] P.M. Pardalos and G.P. Rodgers, Computational aspects of a branch and bound algorithm, Computing 45 (1990), 131–144.
[19] M. Rahman and M. Kuby, A multiobjective model for locating solid waste transfer facilities using an empirical opposition function, INFOR 33 (1995), 34–49.
[20] S.S. Ravi, D.J. Rosenkrantz, and G.K. Tayi, Heuristic and special case algorithms for dispersion problems, Oper Res 42 (1994), 299–310.
[21] C.R. Reeves (Editor), Modern heuristic techniques for combinatorial problems, Halsted, New York, 1993.
[22] L. Schrage, LINDO: An optimization modeling system, Boyd and Fraser, Danvers, 1991.
[23] R.R. Weitz and Lakshminarayanan, An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, VLSI design, and exam scheduling, OMEGA 25 (1997), 473–482.
[24] H.P. Williams, Model building in mathematical programming, Wiley, Chichester, 1993.