ELSEVIER

Operations Research Letters 19 (1996) 175–181

operations research letters

Computational aspects of the maximum diversity problem

Jay B. Ghosh

Faculty of Business Administration, Bilkent University, 06533 Bilkent, Ankara, Turkey

Received 1 November 1994; revised 1 March 1996

Abstract

We address two variations of the maximum diversity problem which arises when m elements are to be selected from an n-element population based on inter-element distances. We study problem complexity and propose randomized greedy heuristics. Performance of the heuristics is tested on a limited basis.

Keywords: Maximum diversity; Computational complexity; Heuristics

1. Introduction

The maximum diversity problem has been addressed off and on in the Operations Research literature. It involves the selection of elements from a population based on measures of overall or worst diversity; the specified inter-element distances usually serve as a surrogate for diversity.

Recently, Kuo et al. [7] have discussed various contexts in which the problem arises such as formulation of immigration and admissions policies, committee formation, curriculum design, market planning and portfolio selection. They have shown that maximizing overall diversity is NP-hard, and have gone on to provide mixed 0–1 linear programming formulations for maximizing both overall and worst diversities. The interested reader is referred to [7] for further details on the maximum diversity problem. In addition, it may be noted that alternative models of diversity, based on considerations that are somewhat different from those of the diversity problems addressed in [7], have been introduced by Glover [5].

In this communication, we restate the diversity problems as treated in [7] and show that maximizing worst diversity is NP-hard as well. We present greedy randomized heuristics for solving two versions of the maximum diversity problem. We also discuss how small instances can be solved exactly via 0–1 quadratic programming, and report computational results to show that our heuristics have performed well.

A couple of points should be made before we proceed. Kuo et al. [7] have mentioned several extensions to the basic diversity problems. One extension involves side constraints that may occasionally warrant consideration. In this regard, we note that our heuristics are quite flexible in their structures and should be able to accommodate such constraints easily. Another extension involves lexicographic maximization of the worst diversity where, in addition to the worst diversity, one wants to maximize the second worst diversity and so on. We note here that, using an approach such as those of Burkard and Rendl [1], our heuristics can be adapted to effectively address this situation as well.

PII S0167-6377(96)00025-9

2. Problem and complexity

Let N be a population of n elements and define d_{ij} to be the specified distance between any two elements i and j. We will assume (without loss of generality) that $d_{ij} = d_{ji} \geq 0$ for all $i, j \in N$ and $d_{ii} = 0$ for all $i \in N$.

In many practical applications, an element i will be characterized by a vector $\langle a_{i1}, \ldots, a_{iq} \rangle$ of q attributes, and d_{ij} will be measured by a metric such as the L_p norm:

$$d_{ij} = \left[\sum_{1 \leq s \leq q} |a_{is} - a_{js}|^p \right]^{1/p}.$$

Now, let M be a subset of N, and assume that the diversity of M can be expressed as a function of d_{ij} for all $i, j \in M$. Suppose that the cardinality of M is restricted to be m. The MAXSUM diversity problem focuses on the overall diversity given by $z(M) = \sum_{i<j;\, i,j \in M} d_{ij}$. The MAXMIN diversity problem, on the other hand, considers the worst diversity $z(M) = \min_{i<j;\, i,j \in M} \{d_{ij}\}$. In both cases, the idea is to maximize $z(M)$ subject to $|M| = m$.

MAXSUM has been studied at length [7]. With a reduction from the clique problem, it has in fact been proved that MAXSUM is strongly NP-hard. Thus, it is very unlikely that MAXSUM will ever be solved in polynomial (or even pseudo-polynomial) time.

Using a reduction from the vertex cover problem [4], we now prove that MAXMIN too is strongly NP-hard. Consider an instance of vertex cover (which is known to be strongly NP-hard): given a graph $G = (V, E)$ and a positive integer $m' \leq |V| = n$, is there a subset V' of V with $|V'| \leq m'$ such that for each edge $(i, j) \in E$ at least one of i and j belongs to V'?

From the above, create an instance of MAXMIN as follows: let a vertex in V correspond to an element in N; for $(i, j) \in E$, let $d_{ij} = 1$; and for $(i, j) \notin E$, let $d_{ij} = 2$. It is easy to see that this transformation is polynomial and that the largest number in the MAXMIN instance is appropriately bounded by a fixed polynomial function of the largest number in the vertex cover instance.

We now show that there is a vertex cover V' of size less than or equal to m' if and only if MAXMIN has a solution M with $z(M) \geq 2$ and $|M| = n - m' = m$.

Suppose that MAXMIN has a solution M. Form $V' = N - M$; note that $|V'| = m'$. Since $z(M) \geq 2$, M does not contain both i and j if $d_{ij} = 1$, that is, if $(i, j) \in E$. Thus, V' is a legitimate vertex cover: it contains at least one of i and j for all $(i, j) \in E$.

Conversely, suppose that MAXMIN does not have a solution: that is, for all M with $|M| \geq m$, $z(M) < 2$. Thus, any legitimate M contains at least one pair of i and j such that $(i, j) \in E$; this implies the absence of a vertex cover V' of size less than or equal to m'.

MAXMIN is clearly in NP. With the above, we have in effect proved that it is strongly NP-hard. Thus, like MAXSUM, it is also computationally difficult.

We note at this point that the MAXMIN instance used in our proof obeys the triangle inequality on the d_{ij}'s. Therefore, MAXMIN remains strongly NP-hard even in this restricted case.

3. Greedy randomized heuristics

Since MAXSUM and MAXMIN are both strongly NP-hard, we focus on their approximate solution. While several heuristic approaches exist for similar problems (see [9]), we turn to a greedy randomized approach. This has of late been used successfully on a number of difficult problems (see, for example, [6, 2, 3]).

Generically, a heuristic of this kind (often called a greedy randomized adaptive search procedure or GRASP) consists of two phases. In the first phase, a solution is iteratively constructed through controlled randomization. In the second, the solution is improved upon through steepest ascent neighborhood search. The process is carried out a number of times and the best solution obtained is delivered as the heuristic solution.

The uniqueness of our particular heuristics derives from the construction and search strategies used in the two phases. We begin with a discussion of the construction phase.

Let M_{k-1} be a partial solution with $k - 1$ $(1 \leq k \leq m)$ elements. For any $i \in N - M_{k-1}$, let $\Delta z(i)$ be the marginal contribution made by i toward

$z(M_m)$. Since the final solution M_m is yet undetermined, we introduce $\Delta z_L(i)$, $\Delta z_U(i)$ and $\Delta z'(i)$ as, respectively, a lower bound, an upper bound and an estimate of $\Delta z(i)$.

Having constructed M_{k-1} iteratively from M_0, we first compute $\Delta z_L(i)$ and $\Delta z_U(i)$ for all $i \in N - M_{k-1}$. Next, a random number u is sampled from a U(0, 1) distribution. This is used to compute $\Delta z'(i) = (1-u)\Delta z_L(i) + u\Delta z_U(i)$. An element i^* is then identified such that $\Delta z'(i^*) = \max_{i \in N - M_{k-1}} \{\Delta z'(i)\}$; i^* is included in M_{k-1} to obtain M_k. This is repeated until M_m is finally delivered.

We now see how $\Delta z_L(i)$ and $\Delta z_U(i)$ are computed. Let $d_i^r(Q_{ik})$ be the rth largest distance in $\{d_{ij}: j \in Q_{ik}\}$, where Q_{ik} is given by $Q_{ik} = N - M_{k-1} - \{i\}$. For MAXSUM, the computations are as follows:

$$\Delta z_L(i) = \sum_{j \in M_{k-1}} d_{ij} + \sum_{n-m+1 \leqslant r \leqslant n-k} d_i^r(Q_{ik});$$

$$\Delta z_U(i) = \sum_{j \in M_{k-1}} d_{ij} + \sum_{1 \leqslant r \leqslant m-k} d_i^r(Q_{ik}).$$

Similarly, for MAXMIN, we have:

$$\Delta z_L(i) = \min\left\{\min_{j \in M_{k-1}} \{d_{ij}\}, d_i^{n-k}(Q_{ik}), z(M_{k-1})\right\} - z(M_{k-1});$$

$$\Delta z_U(i) = \min\left\{\min_{j \in M_{k-1}} \{d_{ij}\}, d_i^{m-k}(Q_{ik}), z(M_{k-1})\right\} - z(M_{k-1}).$$

The search phase begins at the conclusion of the construction phase and attempts to improve upon an incumbent solution through neighborhood search. In this study, we define the neighborhood of a solution to be the set of all solutions obtained by replacing an element in the incumbent solution by another that is not in it. Let M be the incumbent solution. We compute for each $i \in M$ and $j \in N - M$, the improvement due to the exchange of i and j, $\Delta z(i,j)$. If $\Delta z(i,j) \leqslant 0$ for all i and j, then the search is terminated; otherwise, i and j from an i–j pair yielding the maximum $\Delta z(i,j)$ are swapped to obtain a new incumbent solution.

The computation of $\Delta z(i,j)$ is rather straightforward. For MAXSUM, we compute $\Delta z(i,j) = \sum_{u \in M - \{i\}} (d_{ju} - d_{iu})$. Similarly, for MAXMIN, we compute $\Delta z(i,j) = \min_{u<w;\, u,w \in M - \{i\} + \{j\}} \{d_{uw}\} - z(M)$. Note, however, that the computational effort to compute a single $\Delta z(i,j)$ is $O(m)$ for MAXSUM but $O(m^2)$ for MAXMIN.

Each time the two phases are executed to termination, we get a candidate solution. The best solution in a predetermined number of replications (say t) is delivered as the heuristic solution. t is the only parameter in the heuristic that needs tuning. From past experience [3], it is known that a small value such as 10 is usually sufficient. We therefore use $t = 10$ in our computational study.

4. Exact solutions

We test the quality of our heuristic solutions by comparing them against exact solutions for small problem instances (small n and/or m). To obtain exact solutions, we could use the mixed 0–1 linear programming formulations for MAXSUM and MAXMIN [7], and solve them using a commercial solver. We have, however, opted to solve 0–1 quadratic programming formulations using an algorithm due to Pardalos and Rodgers [8]. (A parallel implementation of this algorithm can actually solve quite large 0–1 quadratic programs; see Pardalos et al. [10].) We show below how to cast MAXSUM and MAXMIN as 0–1 quadratic programs.

Let $x_i = 1$ if $i \in N$ is also in M and 0 otherwise. MAXSUM can be modeled as follows:

$$\min_{\{x_i \in \{0,1\}:\, i \in N\}} -\sum_{i<j;\, i,j \in N} d_{ij}x_ix_j + B\left(\sum_{i \in N} x_i - m\right)^2,$$

where B is a large number. Let $d^r(N)$ be the rth largest distance in the set $\{d_{ij}: i<j;\, i,j \in N\}$; then $B = \sum_{1 \leqslant r \leqslant m(m-1)/2} d^r(N)$ can be shown to be sufficiently large for $m > 3$.

MAXMIN can be solved by repeatedly solving a vertex packing problem in a binary search scheme. For a given threshold z, the 0–1 quadratic program for the vertex packing problem is as follows:

$$\min_{\{x_i \in \{0,1\}:\, i \in N\}} -\sum_{i \in N} x_i + C \sum_{(i,j) \in F} x_ix_j,$$

where C is a large number and $F = \{(i,j)\colon d_{ij} < z;\ i < j;\ i,j \in N\}$. For obvious reasons, $C = n$ is deemed sufficiently large. Note that if the 0–1 quadratic program returns a solution value less than $-m$, MAXMIN has a solution M with $z(M) \geqslant z$ and $|M| \geqslant m$; otherwise, it does not have such a solution. Thus, selecting values of z from the set $\{d^r(N)\colon 1 \leqslant r \leqslant n(n-1)/2\}$ in a binary search scheme, we can find in $O(\log n)$ steps (each time solving a 0–1 quadratic program) the maximum value z^* that a solution M with $|M| \geqslant m$ can attain.

5. Computational study

In our basic experiments, five different problem sizes have been explored: $n = 10, 15, 20, 25, 30$. For the MAXMIN problem, additional cases with

Table 1
Computational results for MAXSUM*

Problem size	Measure	Exact solution CPU time (s)	Heuristic solution CPU time (s)	Optimality gap (%)
	Minimum	000.02	<	00.00
$n = 10$	Median	000.02	<	00.00
$m = 2$	Maximum	000.02	<	00.00
	Minimum	000.04	<	00.00
$n = 10$	Median	000.04	000.01	00.00
$m = 4$	Maximum	000.05	000.02	00.00
	Minimum	000.19	000.01	00.00
$n = 15$	Median	000.19	000.01	00.00
$m = 3$	Maximum	000.20	000.02	00.00
	Minimum	001.35	000.02	00.00
$n = 15$	Median	001.35	000.03	00.00
$m = 6$	Maximum	001.36	000.03	00.00
	Minimum	002.47	000.02	00.00
$n = 20$	Median	002.47	000.02	00.00
$m = 4$	Maximum	002.48	000.04	01.13
	Minimum	041.80	000.05	00.00
$n = 20$	Median	042.03	000.06	00.00
$m = 8$	Maximum	042.13	000.06	00.00
	Minimum	032.07	000.04	00.00
$n = 25$	Median	032.17	000.04	00.00
$m = 5$	Maximum	032.27	000.05	00.00
	Minimum	>	000.10	?
$n = 25$	Median	>	000.10	?
$m = 10$	Maximum	>	000.10	?
	Minimum	413.41	000.06	00.00
$n = 30$	Median	413.85	000.07	00.00
$m = 6$	Maximum	415.84	000.08	00.00
	Minimum	>	000.16	?
$n = 30$	Median	>	000.16	?
$m = 12$	Maximum	>	000.18	?

* The symbols "<", ">" and "?", respectively, indicate "less than 000.01 s", "more than 600.00 s" and "unavailable".

$n = 40$ have also been considered. For each size, 5 problem instances have been generated. Note that an instance is completely specified by n, $\{d_{ij}: i < j; i, j \in N\}$ and m. The distances in the set $\{d_{ij}: i < j; i, j \in N\}$ have been sampled from a discrete uniform distribution over [0, 9999], and two different m's – $m = 0.2n$ and $m = 0.4n$ – have been used with each n.

Both the exact algorithms (of which the Pardalos–Rodgers 0–1 quadratic programming solver

Table 2
Computational results for MAXMIN

Problem size	Measure	Exact solution CPU time (s)	Heuristic solution CPU time (s)	Optimality gap (%)
	Minimum	000.02	000.02	00.00
$n = 10$	Median	000.02	000.02	00.00
$m = 2$	Maximum	000.03	000.03	00.00
	Minimum	000.02	000.02	00.00
$n = 10$	Median	000.03	000.03	00.00
$m = 4$	Maximum	000.04	000.04	00.00
	Minimum	000.11	000.06	00.00
$n = 15$	Median	000.11	000.07	00.00
$m = 3$	Maximum	000.12	000.14	00.00
	Minimum	000.11	000.11	00.00
$n = 15$	Median	000.12	000.13	00.00
$m = 6$	Maximum	000.13	000.20	00.00
	Minimum	000.26	000.18	00.00
$n = 20$	Median	000.30	000.22	00.00
$m = 4$	Maximum	000.36	000.28	11.85
	Minimum	000.28	000.33	00.00
$n = 20$	Median	000.31	000.37	00.00
$m = 8$	Maximum	000.36	000.46	01.31
	Minimum	000.54	000.49	00.00
$n = 25$	Median	000.55	000.62	00.00
$m = 5$	Maximum	000.60	000.68	03.13
	Minimum	000.65	000.71	00.00
$n = 25$	Median	000.77	000.88	01.29
$m = 10$	Maximum	000.91	001.30	15.94
	Minimum	001.05	000.86	00.00
$n = 30$	Median	001.25	001.09	00.00
$m = 6$	Maximum	001.50	001.35	07.50
	Minimum	001.95	001.34	00.00
$n = 30$	Median	002.46	002.11	00.00
$m = 12$	Maximum	003.18	002.27	12.98
	Minimum	003.29	002.38	00.00
$n = 40$	Median	005.92	003.96	05.83
$m = 8$	Maximum	006.00	004.68	14.27
	Minimum	010.87	006.38	00.00
$n = 40$	Median	016.68	006.83	12.51
$m = 16$	Maximum	023.77	008.62	22.54

is a part) and the greedy randomized heuristics have been coded in Sun FORTRAN, and all computational runs have been made on a SPARCstation 2 machine operating under SunOS 4.1.1. A CPU time limit of 600 s has been imposed on each run.

Table 1 presents the results of our basic experiments with MAXSUM. The minimum, median and maximum CPU time in seconds taken by the exact and heuristic approaches are shown for each n–m pair. For each such pair, the table also shows the minimum, median and maximum optimality gaps ($=100[z_{\text{exact}} - z_{\text{heuristic}}]/z_{\text{exact}}$). We see that we have been able to solve exactly all 25 instances of MAXSUM with the smaller m in less than 416 s; with the larger m, however, we have been able to solve exactly, within the time limit of 600 s, only the 15 instances for which $n \leqslant 20$. We also see that the heuristic has been extremely effective for the test problems. It has demonstrably found the exact solutions in 39 of the 40 cases where such solutions have been available; in the one case where it has failed, the optimality gap has only been 1.13%. The heuristic has never taken more than 0.18 s.

Table 2, which is organized similar to Table 1, presents our findings on MAXMIN for both the basic and extended experiments. Even though MAXMIN requires the solution of several 0–1 quadratic programs, it has delivered the exact solutions to all 60 instances in less than 24 s. As for the heuristic, we see that it has been reasonably effective. It has found the optimal solutions in 41 of the 60 cases, never taking more than 9 seconds; in the 19 cases where it has failed, the optimality gaps have been less than 23%.

Several observations are in order. First, even though MAXSUM and MAXMIN are both strongly NP-hard, we see that the computational limit of the exact approach for MAXSUM is reached at $n \geqslant 25$ whereas that of the similar approach for MAXMIN extends to $n > 40$. This may not be totatlly surprising since maxsum problems are usually harder to solve than their maxmin counterparts. Next, despite the fact that the heuristic approaches for MAXSUM and MAXMIN are identically structured, the heuristic for MAXMIN is considerably slower than that for MAXSUM. (In fact, the heuristic solution times for MAXMIN are similar to the exact solution times through $n = 25$; the situation begins to change only at $n \geqslant 30$.) The computation of the $\Delta z(i, j)$ may be partially responsible for this. (Recall the computational orders given in Section 3!) An implementation that uses more sophisticated data structures should make the heuristic more efficient. Also, the quality of the heuristic solutions for MAXMIN is noticeably poorer than that for MAXSUM. This may be attributed to the pairwise exchange scheme used in the neighborhood search phase: the MAXMIN heuristic appears to be more vulnerable to being trapped in a local maximum.

Finally, we note that our computational experiments have been performed with the most general instances of the maximum diversity problem. As indicated in Section 2, the d_{ij}'s in many cases will be distances in some metric space and will thus obey the triangle inequality. Even though the problem still remains strongly NP-hard (see Section 2 for the proof in the MAXMIN case), one may conjecture that the computational results will improve over this subset of instances. It will be interesting to see if this is in fact true.

Acknowledgements

Thanks are due to Panos Pardalos and Greg Rodgers for letting us use their unconstrained 0–1 quadratic programming code. Thanks are also due to Jay Rajasekera for helping us with the use of the SPARCstation. The current version of the paper has benefited significantly from the helpful comments of two referees and an associate editor.

References

[1] R.E. Burkard and F. Rendl, "Lexicographic bottleneck problems", *Oper. Res. Lett.* **10**, 303–308 (1991).

[2] T.A. Feo, V. Krishnamurthy and J.F. Bard, "A GRASP for a difficult single machine scheduling problem", *Comput. Oper. Res.* **18**, 635–643 (1991).

[3] T.A. Feo and M.G.C. Resende, "Greedy randomized adaptive search procedures", *J. Global Optim.* **6**, 109–133 (1995).

[4] M.R. Garey and D.S. Johnson, *Computers and Intractability*, W.H. Freeman and Company, New York, 1979.

[5] F. Glover, "Advanced netform models for the maximum diversity problem", Working Paper, Graduate School of Business Administration, University of Colorado at Boulder, Boulder, Colorado, 1991.
[6] J.P. Hart and A.W. Shogan, "Semi-greedy heuristics: an empirical study", *Oper. Res. Lett.* **6**, 107–114 (1987).
[7] C.-C. Kuo, F. Glover and K.S. Dhir, "Analyzing and modeling the maximum diversity problem by zero-one programming", *Dec. Sci.* **24**, 1171–1185 (1993).
[8] P.M. Pardalos and G.P. Rodgers, "Computational aspects of a branch and bound algorithm for quadratic zero-one programming", *Computing* **45**, 131–144 (1990).
[9] P.M. Pardalos and H. Wolkowicz (eds.), *Quadratic Assignment and Related Problems*, DIMACS Series, Vol. 16, American Mathematical Society, 1994.
[10] P.M. Pardalos, A.T. Phillips and J.B. Rosen, *Topics in Parallel Computing in Mathematical Programming*, Science Press, Moscow, 1993.