A Breakpoints Based Method for the Maximum Diversity and Dispersion Problems

Dorit S. Hochbaum* Zhihao Liu† Olivier Goldschmidt‡

Abstract

The maximum diversity, or dispersion, problem (MDP), is to select from a given set a subset of elements of given cardinality (budget), so that the sum of pairwise distances, or utilities, between the selected elements is maximized. We introduce here a method, called the **Breakpoints** (BP) algorithm, based on a technique proposed in Hochbaum (2009), to generate the concave piecewise linear envelope of the solutions to the relaxation of the problem for all values of the budget. The breakpoints in this envelope are provably optimal for the respective budgets and are attained using a parametric cut procedure that is very efficient. The performance of the parametric cut is further enhanced by a newly introduced compact formulation of the problem. The problem is then solved, for any given value of the budget, by applying a greedy-like method to add or subtract nodes from adjacent breakpoints. This greedy solution may be improved using Tabu Search. the BPTS algorithm. This method works well if for the given budget there are breakpoints that are "close". However, for many data sets and budgets this is not the case, and the breakpoints are sparse. We introduce a perturbation technique applied to the utility values in cases where there is paucity of breakpoints, and show that this results in denser collections of breakpoints. Furthermore, each optimal perturbed solution is quite close to an optimal non-perturbed solution. We compare the performance of our breakpoints algorithm to leading methods for these problems: The metaheuristic OBMA–that was recently shown to outperform GRASP, Neighborhood search and Tabu Search–and Gurobi, an integer programming software. It is demonstrated that our method dominates, dramatically, the performance of these methods in terms of computation speed and in comparable or better solution quality. This is the first time that the properties of the concave envelope and the breakpoints are used in a practically tested algorithm. This implies the potential of this method for more general submodular maximization problems.

*Department of IEOR, UC Berkeley, CA, dhochbaum@berkeley.edu

†Department of IEOR, UC Berkeley, CA, zhihao_liu@berkeley.edu

‡Riverside County Office of Education, Riverside, CA, goldoliv@gmail.com

1 Introduction.

We address here the maximum diversity problem (MDP), also known as the maximum dispersion problem. MDP is to select a subset of elements of bounded size, from a given set, so that the sum of pairwise distances, or utilities, between the selected elements is maximized. This problem is NP-hard since it generalizes the maximum clique problem.

The maximum diversity problem (MDP) arose in many different applications such as genetic engineering, transportation system control and alternative energy options [11]. One of its applications that has received a lot of attention recently is the team formation problem in social networks. In this scenario, the cardinality constraint is the maximum headcount budgets and the pairwise distance is the utility of collaboration between each pair of people. The goal is to select people to form a team maximizing the total utility of the team.

The problem can be formalized as a graph problem. Given a graph $G = (V, E)$ with non-negative edge weights u_{ij} for every edge $[i, j] \in E$ and a budget B, the problem is to find a subset $S \subset V$ so that the cardinality of S does not exceed B and so that the sum of weights of edges within S is maximum. Our study here applies to a generalized form of the problem where each node $i \in V$ has a cost q_i, which is permitted to be negative, and the total cost of the set S must satisfy $\sum_{i \in S} q_i \leq B$. We will refer to it as *weighted* MDP.

Since the maximum diversity problem is considered for $q_i = 1$ the discussion and experimentation here will be presented for this case only. But, as noted above, the algorithmic method applies for the weighted MDP as well. In the sequel we will refer to the number of elements $|V|$ as n and the number of pairwise utilities, $|E|$, as m.

MDP is often formulated as a quadratic binary optimization problem. Let x_i be a binary variable which

is equal to 1 if node i is selected in S and 0 otherwise.

$$\begin{aligned} \text{(MDP)}\ \max \quad & \sum_{[i,j]\in E} u_{ij}x_ix_j \\ \text{s.t.} \quad & \sum_{i\in V} x_i \le B \\ & 0 \le x_i \le 1 \text{ integer} \quad \forall\, i \in V, \end{aligned}$$

As noted above the problem is NP-hard and therefore challenging to solve optimally within reasonable amount of time. We consider here the relaxation of the problem's budget constraint for a Lagrange multiplier λ:

$$\begin{aligned} \text{(λ-MDP)}\ \max \quad & \sum_{[i,j]\in E} u_{ij}x_ix_j - \lambda \sum_{i\in V} x_i \\ \text{s.t.} \quad & 0 \le x_i \le 1 \text{ integer} \quad \forall\, i \in V, \end{aligned}$$

As explained next, this relaxed problem λ-MDP is polynomial time solvable, and moreover, one can get the concave envelope describing the solutions for all values of λ in polynomial time [15]. Furthermore, it was shown in [15] that this entire collection of solutions, for all values of λ, is derived in the running time required to solve a minimum cut on an associated graph, using the parametric minimum cut procedure of [14, 16, 17].

Figure 1 illustrates, for each budget the optimal value of the objective function. Obviously, as the budget increases, the value of the optimum can only go up. We then consider a collection of lines that lie above all these points. It suffices to consider the line segment starting at the origin and has lowest slope while still lying above all the points. The lower envelope, which is the minimum of this collection of lines that lie above the collection of these points, for each segment, is then known to be concave. Any point where the line segment changes, and the slope becomes lower, is called a *breakpoint*. We call the slope of the first line segment λ_1, the second λ_2, which is lower in value, etc., for a total of ℓ breakpoints.

It is proved in [15] that the first breakpoint is at the densest subgraph, and λ_1 is the maximum density value.

The statement that we can derive the solutions for all values of λ appears surprising at first glance since the domain of λ values is infinite. As shown in [15], there are at most n different solution sets for all values of λ and they are furthermore nested as the values of λ go down, so all can be represented in $O(n)$ space and time. Details are provided in Section 5.

The concave piecewise linear function that maps all possible budgets to an upper bound on the optimal solution for each respective budget is referred to as the *concave envelope*. The dynamic evolution problem studied in [15] addresses how the solution to MDP evolves as the budget increases. The properties of the concave envelope include:

1. At the breakpoints of this envelope the solutions are optimal.
2. The breakpoints correspond to solutions that are *nested*- the set corresponding to the solution in one breakpoint is a subset of the solutions for larger budgets breakpoints.
3. The number of breakpoints is at most n, the number of elements, or nodes, in the graph G.
4. The envelope describes an upper bound on the optimal value at any level of the budget.
5. If there are optimal solutions that lie on the line segments of the envelope, a method, described in [15], generates such solutions, in constant time per solution.

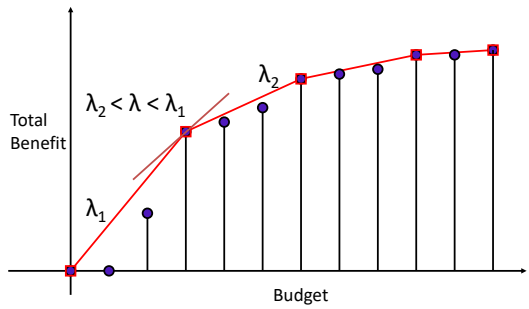


Figure 1: The optimal solutions for each budget, indicated as circles, the concave envelope, in red, and the breakpoints, indicated as squares.

If a breakpoint exists for a given budget value then the problem is solved optimally. In general, however, there are few breakpoints, and, for some problem instances, there could be no breakpoints at all. We develop here insights into the conditions that tend to increase the number of breakpoints, specifically, the larger variety of values from which u_{ij} are generated, the larger is the number of breakpoints. We then develop a *perturbation* technique that increases the number of breakpoints and also affect their distribution. We further show, empirically, that the solutions generated from the perturbed weights are close to the optimal solutions for the non-perturbed weights. Finally, we show that with the breakpoints of the envelope we can compute close to optimal solutions and at much faster running time than state-of-the-art approaches.

Another important contribution here is a new formulation of λ-MDP where the parametric minimum cut is solved on an associated directed graph that is of the same size as G. The associated graph has $n + 2$ nodes

and $2m+n$ arcs. This is in contrast to the known technique that formulates this problem as a *selection problem* on a graph with $O(m)$ nodes and $O(m)$ edges. The difference is computationally significant for the larger datasets.

The paper is organized as follows. We review the relevant literature on MDP in Section 2. Section 3 presents a new compact formulation of λ-MDP and related problems, such as the maximum density subgraph problem. In Section 4 we discuss how to solve λ-MDP for all values of λ in the complexity of a single minimum cut procedure. The BP algorithm that uses a greedy procedure, and its variant BPTS, that is enhanced with Tabu Search, are described in Section 5. Next, the perturbation technique is introduced in Section 6. Our experimental study is using real datasets, existing benchmarks and newly introduced synthetic datasets, that are described in Section 7. The performance of the parametric cut procedure on all the data instances is provided in Section 8, as well as the discussion of the perturbation techniques used for some of the datasets. The empirical study comparing the performance of our algorithm(s) to that of OBMA and Gurobi is presented in Section 9. We conclude with several comments in Section 10.

2 Literature review.

Early applications of MDP arose in the facility location field. Kuby [21] considered maximizing the average distance between selected facilities in the network, calling the problem *the maximum dispersion problem*. Kincaid [20] proposed two metaheuristic algorithms for MDP including simulated annealing and Tabu Search respectively. Kuo et al. [22] proved that MDP is NP-hard even when the distances can assume negative values. They also developed four heuristics: C1, C2, D1 and D2 as the foundation for specialized situations [11]. These heuristics start from an empty set and select one node from unselected nodes at each iterations or start from the collection of all nodes and remove one node at each iterations. Compared to C2 and D2, Ağca et al. [1] proposed a Lagrangian method for MDP and the experimental results in small size data sets (maximum size=100) showed this Lagrangian method had superiority in accuracy but inferiority in the running time.

A different type of application of weighted MDP to text summarization, was discussed by [23]. The authors proposed to use a simple greedy algorithm for the problem, without using information derived from the breakpoints. Our procedure here applies for this text summarization problem with potential improvement of quality of solutions and running times.

Other recent algorithms for MDP have been devised based on the use of Greedy Randomized Adaptive Search Procedure (GRASP) [30,31], Variable Neighborhood Search (VNS) [2, 4, 31] and Memetic Algorithm (MA) [6,34,36]. All these algorithms construct feasible solutions and then try to improve on them by conducting local search for better solutions. Tabu Search [12] is the most widely used local search method for MDP: Palubeck [26] proposed the Iterated Tabu Search (ITS) method in which some elements of the current solution are randomly replaced with unselected elements at each iteration and Tabu Search is then applied. Aringhieri et al. [3] presented the eXploring Tabu Search (XTS) method that maintain, in long-term memory, several solutions in addition to the best incumbent,in order to retain a diverse collection of solutions. Wang et al. [33] proposed a learnable Tabu Search by combining Tabu Search with the Estimation of Distribution Algorithm (EDA). The novelty in this algorithm is the use of a knowledge model called *clustered EDA* to store historical solutions and extract information to guide Tabu Search.

Currently, memetic algorithms tend to outperform other metaheuristics. A memetic algorithm is a combination of population-based search framework and neighborhood-based local search framework. Wang et al. [34] proposed a Tabu memetic algorithm (TS-MA) that in a first stage, applies Tabu search to improve a randomly generated feasible solution. This feasible solution generation is used a number of times, until the collection of improved feasible solutions satisfy a certain quality requirements. This collection of resulting solutions serves as the initial population for the second stage of the algorithm, which applies Tabu search to improve the quality of the solutions. De Freitas et al. [6] developed a memetic self-adaptive evolution strategy (MSES) which mainly applies self-adaptive mutation parameters to generate offspring solutions. Zhou [36] proposed an opposition-based memetic algorithm for MDP called OBMA. Compared to TS-MA, this algorithm searches from both candidate solutions and its opposite solution, and uses a rank-based quality-and-distance pool to guarantee the diversity of solutions in the population. As a result, it is also one of the best performing algorithms so far according to a recently published review [25].

While the majority of algorithms for MDP are heuristics and metaheuristics, there are approaches based on network flow. Witzgall and Saunders [35] addressed this problem in the context of locating postal facilities at maximum dispersion. There the utility of choosing a pair of facilities is the pairwise distance, and there is a cost for each selected facility that is not necessarily equal to 1 (as in MDP). Witzgall and Saunders studied the properties of the concave enve-

lope and gave a non-polynomial algorithm to find the breakpoints. Hochbaum [15] generalized and substantially improved their work both in terms of providing a strongly polynomial parametric cut procedure and in terms of identifying additional structural properties.

3 Compact formulation of λ-MDP

One way of solving the λ-MDP problem is to view it as a *selection problem*. In a selection problem there is a set of elements and a collection of subsets. Each subset is associated with a benefit value, and each element has a cost. The objective is to find a collection of subsets that maximize the net benefit of the subsets minus the cost of the elements in the union of subsets. Here the sets are the edges, each of which is a subset of two elements–its endpoints. And each node is an element associated with a cost. The selection problem is presented as a bipartite graph $B = (V_1 \cup V_2), E)$ with V_1 a set of nodes for each edge in E, and V_2 a set of nodes for each node in V. The bipartite graph B has $n + m$ nodes and each edge-node is connected to its two endpoints nodes for a total of $2m$ arcs. The selection problem is then solved by finding a minimum cut on the $\{s, t\}$ bipartite network constructed by adding to B a source node s connected to each node, ij in V_1, with capacity u_{ij}, and a sink node t that has an arc of capacity λ from each node in V_2. Each arc from V_1 to V_2 has capacity ∞. The source set of a minimum cut consists of the nodes and edges of the subgraph maximizing the objective function of λ-MDP. This graph has $m+n+2$ nodes, and $O(m+n)$ arcs. This graph is illustrated in Figure 2.

We introduce here a compact formulation for solving the λ-MDP problem as a minimum cut on a graph with $n + 2$ nodes and $O(m)$ arcs, resulting in a more efficient method.

For a graph $G = (V, E)$, non-negative edge weights u_{ij}, node weights q_i, and two subsets of nodes, $D_1, D_2 \subseteq V$, we Let $C(D_1, D_2) = \sum_{i\in D_1, j\in D_2 \ , \ [i,j]\in E} u_{ij}$. With this notation, (a generalization of) the λ-MDP problem is to find a subset of nodes S so that $C(S, S) - \lambda \sum_{i\in S} q_i$ is maximized, where $C(S, S) = \sum_{i,j\in S \ and \ [i,j]\in E} u_{ij}$. Next we set $q_i = 1$ although the Lemma applies for general node weights. We denote by d_i the weighted degree of node i in G: $d_i = \sum_{j|[i,j]\in E} u_{ij}$.

LEMMA 3.1. *The λ-MDP problem is equivalent to*

$$\max_{S\subseteq V} \sum_{i\in S} (\frac{1}{2} d_i - \lambda \cdot 1) - \frac{1}{2} C(S, \bar{S}). \tag{3.1}$$

Proof. For any subset of nodes $S \subset V$,

$$d(S) = 2C(S, S) + C(S, \bar{S})).$$

Therefore,

$$C(S, S) - \sum_{i\in S} \lambda \cdot 1 = \frac{1}{2}(d(S) - C(S, \bar{S})) - \sum_{i\in S} \lambda \cdot 1.$$

Maximizing this expression is equivalent to maximizing (3.1). ∎

The problem (3.1) is an instance of the *s-excess* problem, solved as a minimum cut on a graph of the size of G [14]. The general s-excess problem is defined on a directed graph $G = (V, A)$ with non-negative arc capacities and with node weights w_i that can be positive or negative:

$$\text{(s-excess)} \qquad \max_{S\subseteq V} \sum_{i\in S} w_i - C(S, \bar{S}).$$

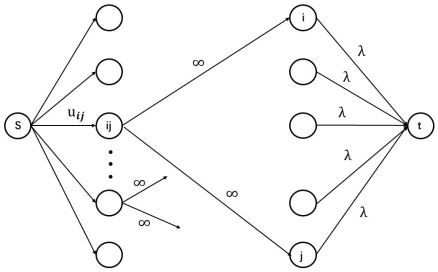


Figure 2: The "selection" graph for λ-MDP

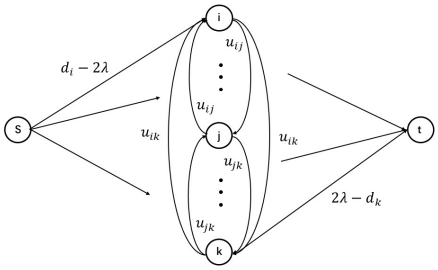


Figure 3: The "s-excess" graph for λ-MDP

Here $w_i = d_i - 2\lambda$. The s-excess problem is a binary special case of a *monotone IP3*: An integer program with at most three variables per inequality, where two of the variables appear with opposite signs and the third variable appear in one constraint at most. It was shown in [13] and [18] that IP3 problems are solved in polynomial time as a minimum cut problem on a respective graph. To show that for our case here, let $z_{ij} = 1$ if $x_i \in S$ and $x_j \in \overline{S}$, and 0 otherwise. With this notation the formulation of 3.1 is,

$$\begin{aligned} \max \quad & \sum_{i\in V} w_i x_i - \sum_{[i,j]\in A} u_{ij} z_{ij} \\ \text{s.t.} \quad & x_i - x_j \le z_{ij} \quad \forall\ (i,j) \in A \\ & 0 \le x_i \le 1 \quad \forall\ i \in V \\ & 0 \le z_{ij} \le 1 \quad \forall\ (i,j) \in A \\ & x_i, z_{ij} \text{ integer} \end{aligned}$$

To solve this problem we construct an s,t-graph $G_{st} = (V \cup \{s,t\}, A \cup A_s \cup A_t)$ as follows: The set of arcs A consists of a pair of opposing directed arcs arcs $(i,j),(j,i)$ for each edge $[i,j] \in E$. Both of these arcs carry the same weight, u_{ij}. The set of arcs A_s go from the source node s to nodes i with $w_i > 0$ and have capacity $u_{s,i} = w_i$. The sink adjacent arcs A_t go from nodes j with $w_j < 0$ to the sink, with capacity $u_{j,t} = |w_j| = -w_j$. The equivalent problem to λ-MDP, (multiplying 3.1 by 2) is

$$\max_{S\subseteq V} \sum_{i\in S} (d_i - 2\lambda) - C(S, \bar{S}).$$

All nodes are connected to the source, s, with capacity $\max\{d_i - 2\lambda, 0\}$, and connected to the sink t, with capacity $-\min\{d_i - 2\lambda, 0\}$.

LEMMA 3.2. *S^* is an optimal solution to λ-MDP defined on graph G if and only if S^* is the source set of a minimum cut in G_{st}.*

Proof. Let $V^+ \equiv \{i \in V | w_i > 0\}$, and let $V^- \equiv \{j \in V | w_j < 0\}$. Let $(s \cup S, t \cup T)$ be a minimum s,t cut on G_{st}. Then the capacity of this cut is given by

$$\begin{aligned} & C\,(s \cup S, t \cup T) \\ = & \sum_{(s,i)\in A_s, i\in T} u_{s,i} + \sum_{(j,t)\in A_t, j\in S} u_{j,t} + \sum_{i\in S, j\in T} u_{ij} \\ = & \sum_{i\in T\cap V^+} w_i + \sum_{j\,\in\, S\cap V^-} -w_j + \sum_{i\in S,\ j\in T} u_{ij} \\ = & \sum_{i\in V^+} w_i - \sum_{i\in S\cap V^+} w_i + \sum_{j\in S\cap V^-} -w_j + \sum_{i\in S, j\in T} u_{ij} \\ = & \ W^+ - \sum_{j\in S} w_j + \sum_{i\in S,\ j\in T} u_{ij} \end{aligned}$$

Where W^+ is the sum of all positive weights in G, which is a constant. Therefore, minimizing $C\,(s \cup S, t \cup T)$ is equivalent to maximizing $\sum_{j\in S} w_j - \sum_{i\in S,\ j\in T} u_{ij}$, and we conclude that the source set of a minimum s,t cut on G_{st} is also a maximum s-excess set of G. ∎

4 Solving λ-MDP for all values of λ: The parametric cut

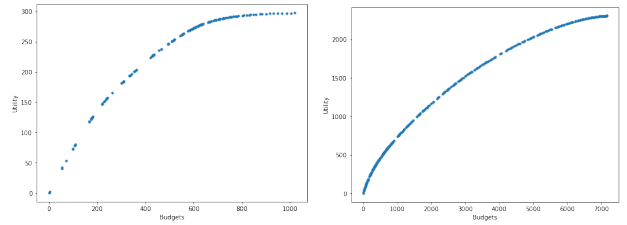
Solving λ-MDP for all values of λ requires to solve the minimum cut problem in Figure 3 for all values of λ. The flow network in Figure 3 is a *parametric* flow network in that the arcs adjacent to the source are monotone non-increasing in the value of λ and the arcs adjacent to the sink are monotone non-decreasing in the value of λ. For a flow network with this property, the maximum flows and minimum cuts for all values of λ can be solved with a *parametric cut* (or parametric flow) procedure in the same complexity as a single minimum cut (or maximum flow). (This is true for parametric functions that are linear, as is the case here, whereas for general monotone parametric functions there is an unavoidable additive factor of $n \log U$ where U is the range for the values of λ). There are only two such parametric cut procedures known. One is based on the push-relabel method [9], and the other is based on the HPF method (Hochbaum's PseudoFlow), in [5, 14, 17].

Let the source set of the minimum cut solving λ-MDP be denoted by S_λ. Then for $\lambda_1 > \lambda_2$ it is known that $S_{\lambda_1} \subseteq S_{\lambda_2}$ - the *nestedness property*. There are at most n, the number of nodes in the graph, values of λ where S_λ changes. Each value of λ in which the solution changes, say λ_j, is called a breakpoint and the solution S_{λ_j} is optimal for λ_j-MDP for budget $B_j = |S_{\lambda_j}|$. Plotting the values of the total utility of S_λ as a function of the cardinality of S_λ, $B = |S_\lambda|$, generates a concave piecewise linear monotone increasing function referred to as the *concave envelope* of the solutions, see Figure 1.

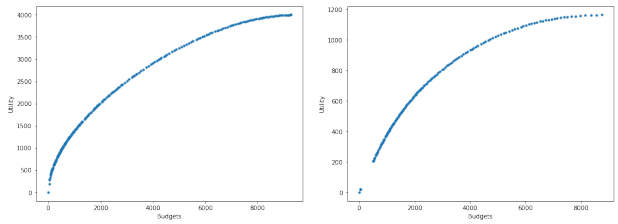
5 The BP algorithm and the concave envelope

We use the guidelines proposed in Hochbaum [15], for utilizing the concave envelope to generate a feasible solution, that is often close to the optimum, for a given budget B. If B corresponds to a breakpoint in the concave envelope, then it is an optimal solution. Otherwise one applies a heuristic, for instance the greedy algorithm, to generate a feasible solution for budget B using the optimal solution at an adjacent breakpoint.

To solve MDP for a budget value B, that does not correspond to a breakpoint the BP algorithm identifies the two adjacent breakpoints to B. That is, let $B_\ell < B_{\ell+1}$, be the budgets of two consecutive breakpoints such that $B_\ell < B < B_{\ell+1}$. The corresponding node sets associated with these breakpoints are $S_\ell \subset S_{\ell+1}$. Let the closest breakpoint to B, in terms of the budget value, be, say, B_ℓ. The greedy algorithm initializes the solutions set S to be equal to S_ℓ. Until $|S| = B$ the following greedy step is repeated. For each node i in $S_{\ell+1} \setminus S$ we calculate the increment of utility

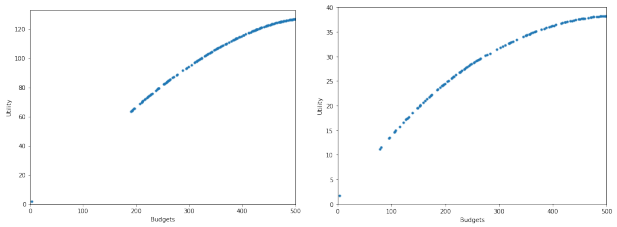


(a) IMDB (b) DBLP

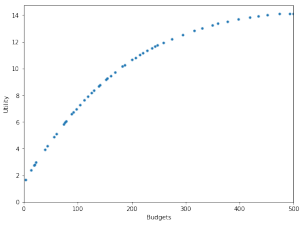


(c) Bibsonomy (d) StackOverFlow

Figure 4: The concave envelopes for the team formation datasets (without perturbation).



(a) $\alpha = 0.8$ (b) $\alpha = 1.0$



(c) $\alpha = 1.2$

Figure 5: Examples of the breakpoint distribution of the perturbed graph for GKD-c_1_n500 dataset with different values of α.

$\Delta_i = \sum_{j \in S} u_{ij}$ that would result by adding node i to S and choose the one $i_{\max}$ that maximizes the value of this increment. For $i_{\max} = \arg\max_{j \in S_{\ell+1} \setminus S} \Delta_j$, we update the set S to $S \leftarrow S \cup i_{\max}$.

Similarly, if the closest breakpoint is on the right, $B_{\ell+1}$, we remove nodes, one at a time, that minimize the loss of utility if removed from the set $S_{\ell+1}$.

The **BPTS algorithm** enhances the greedy feasible solution found by the BP algorithm by applying Tabu Search for several iterations until convergence is achieved.

6 The perturbation technique.

Unlike the example in Figure 4, there are datasets for which there are too few breakpoints, or none at all. This happens if the number of different utility values is small. For instance, if we were to use MDP to solve the maximum clique problem, the utilities would take values of 1 or 0, and in general for such graphs there would be very few breakpoints. This is also the case for some of the datasets we use, that have utility values selected as very small number of integers. To address this, we introduce a perturbation technique. The *exponential* perturbation (see equation 6.2) magnifies the differences between utility values. This perturbation maps the original utility values to the interval $(0, 1]$ while maintaining the monotonicity: if one edge utility is greater than another edge's utility, the perturbed utility of the first is greater than the perturbed utility of the second. The larger the value of utility, the closer the mapped value is to 1 while the smaller the value of utility, the closer the mapped value is to 0. The coefficient α, as it is getting smaller, further magnifies the differences between transformed utility values. Of course the breakpoints generated for the perturbed data are not necessarily optimal for the original utilities. However, we found that in practice, the "pertubed" envelope breakpoints are very close to the actual optimal sets for the original utilities. Examples of the breakpoints distribution for different values of α are shown in Figure 5.

$$u_{ij}^{exp} = e^{-\alpha(u_{\max} - u_{ij})} \tag{6.2}$$

where $u_{\max}$ is the maximu m utility value.

For datasets with utilities assuming small number of integer values, the exponential perturbation is not sufficient. This is the case for the SOM-b datasets (discussed in Section 7). To address that we introduce the *additive perturbation* that modifies the utility values by adding a random number selected uniformly in $(0, 1)$, as in equation 6.3.

$$u_{ij}^{add} = u_{ij} + r; \quad r \sim (0, 1). \tag{6.3}$$

When the additive perturbation is used, it is used first to modify the utility values to u_{ij}^{add} followed by the exponential perturbation.

7 Datasets

The MDPLIB library is a collection of synthetic datasets for maximum diversity problem which are widely used in relevant papers, Martí et al. [24]. We use most of these datasets, excluding those of small size less than 100 nodes. Another benchmark we are using were originally devised for the team formation problem, discussed below. In addition, we introduce here two new benchmarks. One benchmark mimics the maximum diversity datasets, based on real world customer locations in Los Angeles (LA). This benchmark is distinguished in the sizes of the datasets, ranging from 2000 to 6000 nodes, which are much larger than the MDPLIB datasets. A second benchmark introduced here is a collection of ten synthetic datasets designed, as described below, to mimic the team formation datasets. All the parameters of the datasets used are provided in Table 1.

Dataset	# nodes	# edges	#I*	Utility type
GKD-c_n500	500	124750	20	
GKD-d_n500	500	124750	10	Euclidean
GKD-d_n1000	1000	499500	10	distance
GKD-d_n2000	2000	1999000	10	
MDG-b_n500	500	124750	20	Reals
MDG-b_n2000	2000	1999000	20	∼U(0,1000)
SOM-b_n100	100	4950	4	
SOM-b_n200	200	19900	4	
SOM-b_n300	300	44850	4	Integers
SOM-b_n400	400	79800	4	∼U[0,9]
SOM-b_n500	500	124750	4	
LA_n2000	2000	64845	1	
LA_n3000	3000	139645	1	Euclidean
LA_n5000	5000	366301	1	distance
LA_n6000	6000	564617	1	
IMDB	1021	22448	1	
DBLP	7159	30562	1	
Bibsonomy	9271	61422	1	Jaccard
StackOverFlow	8834	124554	1	Similarity
SyntheticTF	7000	36525.8 (mean)	10	

Table 1: The datasets details.
* #I is the number of instances.

The MDPLIB datasets are of three types: GKD, MDG and SOM. They are all complete graphs, that is for each pair of nodes there is an edge with positive utility. They differ in the way the utilities are computed.

The LA datasets were generated by randomly selecting locations from over a million addresses in Los Angeles (for details, see [29]). The pairwise-utilities for these datasets are the Euclidean distances between the respective pairs of locations. Due to the large sizes of these datasets, 2000, 3000, 4000, 5000, and 6000 nodes respectively, we retain only the largest 5% of the utilities.

The team formation datasets are four real data sets used as benchmark to test methods for the team formation problem: IMDB, DBLP, Bibsonomy and StackOverFlow. The team formation problem is to find a team of experts maximizing the collaboration potential of the team while satisfying extra constraints on required skills. The input to the team formation problem is a set of participants, each associated with projects they worked on, and the projects on which they collaborated with others in the past. Let P_i denote the set of person i's projects, P_j denote the set of person j's projects, then the communication between a pair i and j is evaluated by the Jaccard similarity: (see e.g. [8,32]) $J(i,j) = \frac{|P_i \cap P_j|}{|P_i \cup P_j|}$. The studied team formation problem has additional constraints that require to include a coverage of a set of skills by the team. Without this requirement, which we relax, we attain MDP problem instances.

We generate a benchmark of synthetic datasets, *syntheticTF*, that mimics the construction of the team formation benchmark. The datasets are generated as follows: The input parameters are the number of participants, the number of projects, and the mean and standard deviation of a lognormal distribution. With these parameters the construction of syntheticTF is as follows:

1. The set of projects is partitioned into subsets,the cardinality of which is randomly generated, one by one, from the lognormal distribution. For the last subset, if the lognormal value is greater than the remaining number of projects then its size is the remaining number.
2. A number of projects n_i to associate with each participant i is randomly selected from the same lognormal distribution as above.
3. For each participant, choose randomly and uniformly one of the subsets in the partition. If the size of this subset is less than or equal n_i, then choose randomly and uniformly n_i projects from the selected subset. Otherwise, assign the entire subset of projects to participant i and add additional projects to complete the number to n_i by choosing randomly and uniformly among all the remaining projects.
4. The pairwise utility for each pair of participants is computed using the Jaccard similarity.

In total, we generated ten syntheticTF datasets with $7,000$ participants, $70,000$ projects, and a lognormal distribution with mean equal to 4 and standard deviation equal to 1.

Setting budget values. Most empirical results for the MDP problem are run on data instances with one specific budget value. In general, the problem is easier for larger budget values than it is for small budget values. We provide, for each instance, a collection of budget values that depend on the size of the instance. For all instances, other than syntheticTF, we set the largest budget value equal to 40% of the total number of nodes in the instance. Then we set the smallest budget value and increments as follows: For GKD-c and GKD-d data sets with 500 nodes, we set the budget values starting at 20 and increment them by 20; For GKD-d and LA datasets with more than 500 nodes, we set the smallest budget value to 50 with increments of 50; The IMDB dataset has the same budget settings as GKD-c and GKD-d datasets; The other team formation instances have smallest budget set to 100 and increments of 100.

8 The parametric cut and perturbation

In order to identify the concave envelope and respective breakpoints we run the parametric cut algorithm that is based on the HPF (Hochbaum's PseudoFlow) algorithm, [16, 17]. The average running time per instance of each benchmark is provided in Table 2. Even for the largest instances, the running time is below 2 seconds.

Dataset (# instances)	para CPU	Dataset (# instances)	para CPU
IMDB(1)	0.17	GKD-c_n500(20)	0.26
DBLP(1)	1.15	GKD-d_n500(10)	0.22
Bibsonomy(1)	1.53	GKD-d_n1000(10)	0.6
StackOverFlow(1)	1.48	GKD-d_n2000(10)	1.76
SyntheticTF(10)	1.7	MDG-b_n500(20)	0.22
LA_n2000(1)	0.4	MDG-b_n2000(20)	1.83
LA_n3000(1)	0.64	SOM-b_n100(4)	0.03
LA_n4000(1)	0.93	SOM-b_n200(4)	0.08
LA_n5000(1)	1.17	SOM-b_n300(4)	0.12
LA_n6000(1)	1.52	SOM-b_n400(4)	0.16
		SOM-b_n500(4)	0.21

Table 2: The parametric cut (**para**) average CPU time (sec) per instance.

The next issue to determine is the use of the perturbation. A priori there is no information about whether to use perturbation, and if so, which value of α to select in the exponential perturbation. Our experience indicates that smaller values of α increase the overall number of breakpoints, yet larger values of α provide a better spread of the breakpoints for low budgets. An example illustrating this effect is shown in Figure 5 for an instance of the GKD dataset. Some datasets require no perturbation. This is the case for all the team formation datasets including the syntheticTF datasets. The breakpoints graph for the four team formation datasets are shown in Figure 4.

Recall that the datasets SOM-b have utility weights that assume an integer value in $[0, 9]$. This is particularly challenging for the use of the breakpoints since typically there would be no breakpoint other than the two corresponding to the empty set and the entire graph. For these datasets we utilize first the additive perturbation, followed by the exponential perturbation. This perturbation is successful in terms of the quality of the results provided by the BPTS algorithm, and discussed in detail in Subsection 9.

9 Experimental results

We compare the performance of our breakpoints algorithm to leading methods for these problems: The metaheuristic OBMA, that was shown recently to perform better than GRASP, Neighborhood search and Tabu Search, and Gurobi–an integer programming software.

We measure the performance of each algorithm in terms of the value of the objective function derived and the CPU running time. For Gurobi we gave a time limit of 20 seconds for all instances. Comparing running times with OBMA is not quite reflective of the actual work since OBMA is compiled in C++ whereas the greedy and Tabu Search parts of the BPTS code are compiled in Python. To make the comparison between BPTS and OBMA fair, we compare the count of the number of Tabu Search iterations used by both algorithms. Although the running time of BPTS algorithm includes the greedy procedure in addition to the Tabu Search, this portion of the running time is trivial–considerably less than a fraction of a second, even on large data sets. These running times are provided in Table 3 for all datasets except for team formation instances. For the team formation instances, including the syntheticTF datasets, our algorithm is BP which does not employ Tabu Search. For these instances we provide the actual running times comparisons in Figure 7 and Table 5.

Dataset (# instances)	BP CPU	Dataset (# instances)	BP CPU
GKD-c-500(20)	0.055	GKD-d-500(10)	0.038
GKD-d-1000(10)	0.054	GKD-d-2000(10)	0.086
MDG-b-n500(20)	0.062	MDG-b-2000(20)	0.082
SOM-b(20)	0.014	LA(5)	0.072

Table 3: Average CPU time (sec) per instance and per budget for BP(Greedy)

The comparison of the number of iterations of the Tabu search procedure used by OBMA and by BPTS is shown in Figure 6. Note that we apply the same stopping rule in both algorithms: Twenty consecutive iterations without improvement in the value of the solution lead to termination.

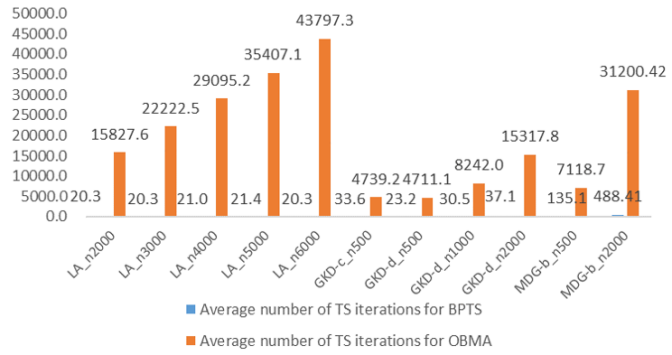


Figure 6: The average number of Tabu Search iterations per budget in BPTS and OBMA.

Objective value comparisons.
To compare the performance of the algorithms we consider the objective value (total utility) of the solutions obtained. Recall that MDP is a maximization problem. Therefore we consider the difference between the objective value of the BPTS solution (or in some of the datasets we run only the BP algorithm) to that delivered by OBMA and by Gurobi. With some abuse of notation, we write "BPTS-OBMA Gap (%)" the incremental difference of the value of the BPTS compared to OBMA, divided by the value of OBMA, and then multiply by 100 to express this difference as a percentage. The same formula is applied for "BPTS-Gurobi Gap (%)". Positive percentages imply that BPTS (or BP) is better whereas negative indicate it is doing worse.

In Table 4 we summarize the relative performance of BPTS (for the first five instances we ran BP) versus that of OBMA and Gurobi.

We first discuss the comparison with OBMA. OBMA is doing extremely badly on all team formation instances. It only offers a small improvement, a fraction of a percent, as compared to BPTS for the two MDG benchmarks, and the instances of SOM. Recall that the utility values of the SOM instances are particularly challenging for the BP algorithm, as they assume an integer value in $[0, 9]$ and yet BPTS's performance is comparable to that of OBMA. In terms of running times, as discussed above, we either compare the number of Tabu Search iterations, in Figure 6, or show the actual running times for the syntheticTF instances, in Table 5.

Dataset	(#B, #I)*	BPTS-OBMA Gap (%)	BPTS-Gurobi Gap (%)
IMDB**	(8,1)	0.69	-0.09
DBLP**	(60,1)	117.71	-0.11
Bibsonomy**	(80,1)	108.27	-0.06
StackOverFlow**	(64,1)	138.26	0.13
SyntheticTF**	(20,1)	7.36	-0.05
LA_n2000	(16,1)	0	0
LA_n3000	(24,1)	0	0.05
LA_n4000	(32,1)	0	0.29
LA_n5000	(20,1)	0	0.24
LA_n6000	(24,1)	0	0.41
GKD-c_n500	(10,20)	0	0.13
GKD-d_n500	(10,10)	0	0.25
GKD-d_n1000	(20,10)	0	1.23
GKD-d_n2000	(16,10)	0	29.2
MDG-b_n500	(10,20)	-0.48	0.87
MDG-b_n2000	(16,20)	-0.28	7.62
SOM-b_n100	(2, 4)	-0.10	0.32
SOM-b_n200	(4, 4)	-0.08	0.83
SOM-b_n300	(6, 4)	-0.18	1.21
SOM-b_n400	(8, 4)	-0.15	1.91
SOM-b_n500	(10, 4)	-0.17	2.23

Table 4: The average per instance and per budget gaps (%) between BPTS objective value and that of OBMA, and Gurobi. Positive percentage means that BPTS performs better.
* #B is the number of budget values for each instance of the given data set; #I is the number of instances of the given data set.
** The gaps are for BP, not BPTS, versus OBMA and Gurobi.

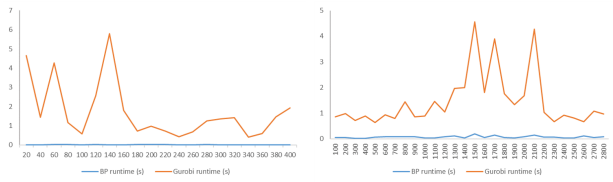
These results indicate that the running times of OBMA are overwhelmingly larger than those of BPTS, or BP.

We next discuss the comparison with Gurobi. In terms of the objective values attained by the Gurobi, these are slightly better than those of BPTS (or BP) for the team formation datasets, but by less than 0.11%. On the other datasets BPTS is doing better and for some instances a lot better. In terms of running time of Gurobi compared to BPTS or BP, we show in Figure 7 the running times of Gurobi versus BP for each budget value of the team formation instances. As can be seen in that figure, the running time of Gurobi is very erratic and varies a lot between different budget values. Furthermore, the running times of Gurobi are dramatically larger than those of BP. Table 6 shows the average running time of Gurobi for the datasets

other than the team formation. Recall that Gurobi has a running time limit of 20 seconds for every dataset and every budget which explains why the running times reported in Table 6 are at most 20+ seconds.

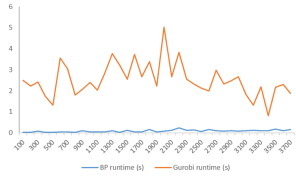
Budget	Average BP-OBMA Gap(%)	Average BP-Gurobi Gap(%)	Average BP CPU	Average OBMA CPU	Average Gurobi CPU
50	20.11	-0.25	0.00	6.93	1.04
100	48.33	-0.36	0.00	10.00	1.02
150	64.89	-0.13	0.00	10.15	1.12
200	69.95	-0.14	0.01	10.31	1.33
250	70.42	-0.14	0.01	10.45	1.31
300	75.61	-0.04	0.02	10.59	1.32
350	70.40	-0.07	0.02	10.75	1.43
400	65.87	-0.07	0.03	10.93	1.44
450	63.57	-0.02	0.04	11.11	1.55
500	63.71	-0.02	0.05	11.29	1.93
550	55.56	-0.04	0.06	11.49	2.38
600	52.60	-0.01	0.07	11.72	2.38
650	58.85	-0.02	0.08	11.95	2.42
700	67.89	-0.04	0.10	12.21	2.76
750	69.46	-0.02	0.11	12.48	3.48
800	84.01	-0.01	0.12	12.71	3.22
850	84.22	-0.01	0.14	13.02	3.92
900	103.72	0.00	0.16	13.34	3.99
950	107.56	-0.01	0.18	13.59	4.20
1000	113.31	-0.01	0.20	14.01	4.11

Table 5: Comparison of BP, OBMA, and Gurobi on the ten syntheticTF instances. The average gap and CPU are per instance.

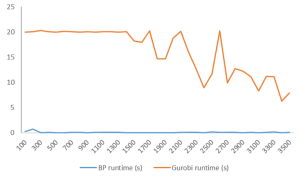


(a) IMDB

(b) DBLP



(c) Bibsonomy



(d) StackOverFlow

Figure 7: Runtime comparison of BP and Gurobi on team formation data sets for all budgets.

Dataset (# instances)	Guro CPU	Dataset (# instances)	Guro CPU
GKD-c-500(20)	20.04	GKD-d-500(10)	20.04
GKD-d-1000(10)	20.19	GKD-d-2000(10)	20.94
MDG-b-n500(20)	20.04	MDG-b-2000(20)	20.40
SOM-b(20)	10.02	LA(5)	13.08

Table 6: Gurobi's (Guro) average CPU time (sec) per instance and per budget.

10 Conclusions

We introduce here, for the first time, an algorithm that uses the breakpoints in the concave envelope of solutions to the parametric relaxation of the problem. This algorithm's performance is illustrated here for solving the maximum diversity problem, MDP, but it also applies for more general problems, including the weighted MDP. The breakpoints are attained efficiently by a parametric cut procedure, on a compact formulation of the problem, introduced here. That formulation reduces the size of the graph from quadratic number of nodes, in the size of the dataset, to a linear number of nodes. A further novelty here is the employment of a perturbation technique on the utility values that results in generating a denser distribution of breakpoints.

An empirical study comparing our algorithm to the leading algorithms OBMA and Gurobi, demonstrates that it provides give comparable or higher quality solutions to the MDP problem in significantly faster running times. Future research includes applying the breakpoints based algorithm to more general problems such as the weighted MDP for text summarization.

Acknowledgement

The first author was supported in part by AI institute NSF award 2112533.

References

[1] Ş. Ağca, B. Eksioglu, and J. B. Ghosh, *Lagrangian solution of maximum dispersion problems*, Naval Research Logistics (NRL), 47 (2000), pp. 97–114.

[2] R. Aringhieri and R. Cordone, *Comparing local search metaheuristics for the maximum diversity problem*, Journal of the Operational research Society, 62 (2011), pp. 266–280.

[3] R. Aringhieri, R. Cordone, and Y. Melzani, *Tabu search versus grasp for the maximum diversity problem*, 4OR, 6 (2008), pp. 45–60.

[4] J. Brimberg, N. Mladenović, D. Urošević, and E. Ngai, *Variable neighborhood search for the heaviest k-subgraph*, Computers & Operations Research, 36 (2009), pp. 2885–2891.

[5] B. G. Chandran and D. S. Hochbaum, *A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem*, Operations research, 57 (2009), pp. 358–376.

[6] A. R. R. de Freitas, F. G. Guimarães, R. C. Pedrosa Silva, and M. J. F. Souza, *Memetic self-adaptive evolution strategies applied to the maximum diversity problem*, Optimization Letters, 8 (2014), pp. 705–714.

[7] A. Duarte and R. Martí, *Tabu search and grasp for the maximum diversity problem*, European Journal of Operational Research, 178 (2007), pp. 71–84.

[8] S. H. Farnoush Farhadi, Maryam Sorkhi and A. Hamzeh, *An effective expert team formation in social networks based on skill grading*, in 2011 IEEE 11th International Conference on Data Mining Workshops, IEEE, 2011.

[9] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, *A fast parametric maximum flow algorithm and applications*, SIAM Journal on Computing, 18 (1989), pp. 30–55.

[10] J. B. Ghosh, *Computational aspects of the maximum diversity problem*, Operations research letters, 19 (1996), pp. 175–181.

[11] F. Glover, C.-C. Kuo, and K. S. Dhir, *Heuristic algorithms for the maximum diversity problem*, Journal of information and Optimization Sciences, 19 (1998), pp. 109–132.

[12] F. Glover and M. Laguna, *Tabu search*, in Handbook of combinatorial optimization, Springer, 1998, pp. 2093–2229.

[13] D. S. Hochbaum, *Solving integer programs over monotone inequalities in three variables: A framework for half integrality and good approximations*, European Journal of Operational Research, 140 (2002), pp. 291–321.

[14] ———, *The pseudoflow algorithm: A new algorithm for the maximum-flow problem*, Operations research, 56 (2008), pp. 992–1009.

[15] ———, *Dynamic evolution of economically preferred facilities*, European Journal of Operational Research, 193 (2009), pp. 649–659.

[16] ———, *Hpf-hochbaum's pseudoflow*, 2020. Accessed: May 28, 2022, https://riot.ieor.berkeley.edu/Applications/Pseudoflow/maxflow.html.

[17] ———, *Pseudoflow parametric maximum flow solver version 1.0*, 2020. Accessed: May 28, 2022, https://riot.ieor.berkeley.edu/Applications/Pseudoflow/parametric.html.

[18] ———, *Applications and efficient algorithms for integer programming problems on monotone constraints*, Networks, 77 (2021), pp. 21–49.

[19] D. S. Hochbaum and A. Pathria, *Analysis of the greedy approach in problems of maximum k-coverage*, Naval Research Logistics (NRL), 45 (1998), pp. 615–627.

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

[21] M. J. Kuby, *Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems*, Geographical Analysis, 19 (1987), pp. 315–329.

[22] C.-C. Kuo, F. Glover, and K. S. Dhir, *Analyzing and modeling the maximum diversity problem by zero-one programming*, Decision Sciences, 24 (1993), pp. 1171–1185.

[23] H. Lin and J. Bilmes, *Multi-document summarization via budgeted maximization of submodular functions*, in Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, 2010, pp. 912–920.

[24] D. Martí and Martínez-Gavara, *The mdplib 2.0 library of benchmark instances for diversity problems*, 2021. Accessed: October 25, 2022, https://www.uv.es/rmarti/paper/mdp.html.

[25] R. Martí, A. Martínez-Gavara, S. Pérez-Peló, and J. Sánchez-Oro, *A review on discrete diversity and dispersion maximization from an or perspective*, European Journal of Operational Research, (2021).

[26] G. Palubeckis, *Iterated tabu search for the maximum diversity problem*, Applied Mathematics and Computation, 189 (2007), pp. 371–383.

[27] F. Parreño, R. Álvarez-Valdés, and R. Martí, *Measuring diversity. a review and an empirical analysis*, European Journal of Operational Research, 289 (2021), pp. 515–532.

[28] J. Peiró, I. Jiménez, J. Laguardia, and R. Martí, *Heuristics for the capacitated dispersion problem*, International transactions in operational research, 28 (2021), pp. 119–141.

[29] D. S. H. Roberto Asín-Achá, Olivier Goldschmidt and I. I. Huerta, *Fast algorithms for the capacitated vehicle routing problem using machine learning selection of algorithm's parameters*, in Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, 2022, pp. 29–39.

[30] G. C. Silva, M. R. De Andrade, L. S. Ochi, S. L. Martins, and A. Plastino, *New heuristics for the maximum diversity problem*, Journal of Heuristics, 13 (2007), pp. 315–336.

[31] G. C. Silva, L. S. Ochi, and S. L. Martins, *Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem*, in International Workshop on Experimental and Efficient Algorithms, Springer, 2004, pp. 498–512.

[32] G. Süer and M. Ortega, *A machine level based-similarity coefficient for forming manufacturing cells*, Computers Industrial Engineering, 1–4 (1994), pp. 67–70.

[33] J. Wang, Y. Zhou, Y. Cai, and J. Yin, *Learnable tabu search guided by estimation of distribution for maximum diversity problems*, Soft Computing, 16 (2012), pp. 711–728.

[34] Y. Wang, J.-K. Hao, F. Glover, and Z. Lü, *A tabu*

search based memetic algorithm for the maximum diversity problem, Engineering Applications of Artificial Intelligence, 27 (2014), pp. 103–114.

[35] D. Witzgall and R. Saunders, *Electronic mail and the "locator's" dilemma*, Applications of discrete mathematics, 33 (1988), pp. 65–84.

[36] Y. Zhou, J.-K. Hao, and B. Duval, *Opposition-based memetic search for the maximum diversity problem*, IEEE Transactions on Evolutionary Computation, 21 (2017), pp. 731–745.