Erasmus University Rotterdam
Erasmus School of Economics

Bachelor Thesis Econometrics and Operations Research

The Maximum Coverage Problem

Investigating Approximation Algorithms for Variants of the Maximum Coverage Problem

Ymro Nils Hoogendoorn (413127)

Supervisor: ir. drs. R. Kerkkamp
Second Assessor: dr. W. van den Heuvel

July 12, 2017

Abstract

In this thesis, we investigate several heuristics to solve the maximum coverage problems and its variants. We consider maximum coverage problems with cardinality, matroid and budget constraints. For each of these problem variants, we solve randomly generated instances with greedy and local search heuristics. In particular, we investigate the actual performances of these algorithms compared to their theoretical approximation ratios, if these exist and are known. We also investigate the performance of a tabu local search algorithm. For the budgeted constrained maximum coverage problem, we construct two relaxation-based heuristics. One uses the principles of subgradient optimization and Lagrange relaxation and the other uses a relaxation-based tabu local search. We find that the non-oblivious swap local search introduced in Filmus and Ward (2012) performs the worst on average of the algorithms, despite having an the best approximation ratio of $(1 - \frac{1}{e})$ on the cardinality- and matroid-constrained maximum coverage problem (unless P=NP). The tabu local search algorithm performs best in these cases, but at the cost of a longer runtime. For the budgeted maximum coverage problem, the relaxation-based tabu swap local search algorithm performs best, but has a longer runtime than existing greedy and local search algorithms.

Contents

1 Introduction

The maximum coverage problem (MCP) is a frequently occurring problem in combinatorics. Suppose we are given a collection of weighted elements and a collection of sets of these elements. The goal of the maximum coverage problem is to select a limited number of sets, such that the total weight of the covered elements in these chosen sets is maximal. A straightforward interpretation of this problem is the so-called cell phone towers interpretation. In this interpretation, the elements represents customers and their weight is the associated profit of covering that customer. A set represents a specific cell phone tower and the customers it can reach. Solving the maximum coverage problem answers the question of how to build a limited number of cell phone towers, while maximizing the profits.

However, the MCP is NP-hard, which means that solving this problem to optimality is often not an option for large instances. Therefore, it is of interest to investigate heuristics to solve this problem. Even though heuristics do not guarantee to find the optimal solution, careful choice of the heuristics can lead to good or even near-optimal solutions. The concept of α-approximation algorithms ($0 \leq \alpha \leq 1$) has been studied thoroughly in the history of the MCP. An α-approximation algorithm guarantees that the returned objective value is at least α (the approximation ratio) times the optimal objective value. In other words, these algorithms guarantee a certain performance, which is why α-approximation algorithms have been the subject of many studies.

Even though α-approximation algorithms guarantee a certain performance, this guarantee does not say anything about the algorithm's actual performance. In particular, we are interested in the performance ratio of a heuristic, which is defined as the ratio of the heuristic solution found to the optimal solution. This thesis will investigate several approximation algorithms for different maximum coverage instances and compare their actual performance against each other and their theoretical performance. We will also investigate some cases in which the algorithms do not have an approximation ratio and some heuristics without an approximation ratio.

To accomplish this task, we record the performance of several algorithms on a few variants of the maximum coverage problem. In particular, we focus our attention on three variants: the previously described maximum coverage problem (MCP), the maximum coverage problem over a matroid (MCPM) and the maximum coverage problem with budget constraints (MCPBC). The three variants only differ in the constraints they impose on solutions. The MCP, as mentioned before, restricts the maximum number of chosen sets. The MCPM restricts solutions to lie in a matroid, which is a special kind of structure. We will elaborate on the precise definition of a matroid in Section 3.4. The MCPBC assigns a nonnegative cost to every set and solutions are feasible only if the total costs of the chosen sets does not exceed a predefined budget.

The algorithms we are going to investigate fall into two categories: greedy algorithms and local search methods. Greedy algorithms, in general, construct solutions by iteratively adding sets that maximize a certain criteria. Local search algorithms, on the other hand, start with a solution and iteratively exchange sets in the current solution with different sets. As already much research has been done on these types of algorithms, we give a overview of the relevant literature of the maximum coverage problem in Section 2.

The main goal of this thesis is to investigate the performance of several greedy and local search algorithms for the three variants of the maximum coverage problem. For the MCPBC, we also construct two additional local search methods. One method is based on the Lagrange relaxation and the other on the principle of tabu search.

The outline of this thesis is as follows. Section 2 gives an overview of the relevant literature of the maximum coverage problems and its variants. This section also describes the principles of Lagrange relaxation and tabu search. In Section 3 we define the maximum coverage problem and its variants formally and introduce the notation that will be used throughout this thesis. Section 4 describes the solution methods that will be investigated. We present in Section 5 the performances of the investigated algorithms and compare these. Finally, Section 6 summarizes these results and suggests further research topics.

2 Literature

The maximum coverage problem (MCP) is a well-studied problem in combinatorics and has a rich history of approximation algorithms and generalizations of the problem. An early paper to define and investigate this problem was Hochbaum and Pathria (1998). This paper investigated the standard greedy heuristic. This heuristic constructs a solution to the problem by iteratively adding sets to the solution, such that the weight increase of adding the set is maximal. They proved that this heuristic achieves an approximation ratio of $(1 - \frac{1}{e})$. In other words, the greedy heuristic always finds an objective value of $(1 - \frac{1}{e})$ times the optimal objective value or higher.

Nemhauser, Wolsey, and Fisher (1978) considered a more general version of the problem: maximization of a monotone submodular function under a cardinality constraint. Submodular functions are functions that assign a real value to a set. They also adhere to a diminishing returns principle, and can be seen as a discrete equivalent to concave functions. Monotone submodular functions have the additional property that larger sets yield higher or equal function values. The objective function (weight) of the maximum coverage problem is also a monotone submodular function. Nemhauser et al. (1978) found the same result as Hochbaum and Pathria (1998), but several year earlier. That is, the greedy algorithm achieves an approximation ratio of $(1 - \frac{1}{e})$ even in this more general case. It was already known that the maximum coverage problem is NP-hard, but Feige (1998) proved the inapproximability of the problem. That is, unless P=NP, it is impossible for an algorithm to achieve an approximation ratio strictly greater than $(1 - \frac{1}{e})$. This also implies that the greedy algorithm actually achieves the highest possible approximation ratio of the MCP.

A different approach of solving maximum coverage problem instances was investigated by Resende (1998). This paper used a greedy randomized adaptive search procedure (GRASP) as a heuristic. Furthermore, he obtained upper bounds for the optimal weight by considering a linear programming relaxation. It was shown that for all random instances considered by Resende (1998), GRASP performed better than the greedy algorithm and the solutions found were nearly optimal. However, unlike the greedy algorithm, no theoretical performance guarantee was proven for GRASP.

One year later, Khuller, Moss, and Naor (1999) investigated an extension to the maximum coverage problem: the maximum coverage problem with budget constraints (MCPBC). Instead of restricting the maximum number of sets in a solution, each set is given a costs and solutions are feasible only if the total cost of the chosen sets is smaller than some predefined fixed budget. First they investigated a greedy-like algorithm very similar to the standard greedy algorithm. The only difference is that this greedy-like algorithm prefers sets with the highest weight increase to cost ratio, instead of just using the weight increase. As they showed that this greedy-like algorithm in this setting does not have an approximation ratio, Khuller et al. (1999) devised two algorithms. The first algorithm calculated the weight for all solutions with one set. Then, the first algorithm constructs another set using the aforementioned greedy-like algorithm. The first algorithm then returns the solution with the greatest weight of all encountered ones. This algorithm achieves an approximation ratio of $\frac{1}{2}(1 - \frac{1}{e})$. The second algorithm actually calculates the weight all solutions containing strictly less than κ sets. Then, it uses the same greedy-like approach as the first algorithm on all solutions containing exactly κ sets. The algorithm returns the encountered solution with the greatest weight. This achieves an approximation ratio of $(1 - \frac{1}{e})$ if $\kappa \geq 3$. Furthermore, Khuller et al. (1999) showed that $(1 - \frac{1}{e})$ is the highest approximation ratio possible for the MCPBC under similar conditions given by Feige (1998).

Continuing the study of greedy algorithms, Chekuri and Kumar (2004) studied the performance of the greedy algorithm on the maximum coverage problem with group budget constraints. This variant is also known as the maximum coverage problem with a partition matroid constraint. The problem has the available sets partitioned into predefined groups, where a feasible solution is only allowed to contain a certain number of sets per partition. Chekuri and Kumar (2004) showed that the standard greedy algorithm for this variant only achieves an approximation ratio of $\frac{1}{2}$.

Just like Nemhauser et al. (1978), Calinescu, Chekuri, Pál, and Vondrák (2011) considered maximization of a monotone submodular function. However, Calinescu et al. (2011) subjected the monotone submodular

function to a matroid constraint, which is broader than the cardinality constraint in Nemhauser et al. (1978). A matroid is essentially a special structure and can be seen as a collection of subsets satisfying some rules. The group budget constraint (partition matroid) considered in Chekuri and Kumar (2004) is a special case of a matroid constraint, thus the analysis of Calinescu et al. (2011) can be applied to many of the previously considered variants of the maximum coverage problem. Calinescu et al. (2011) proved that the standard greedy algorithm achieves an approximation ratio of $\frac{1}{2}$ in this general case, which is the same approximation ratio Chekuri and Kumar (2004) found for the partition matroid. Calinescu et al. (2011) used a continuous greedy-like process with pipage rounding. The algorithm first constructs a solution using a continuous greedy-like process, allowing a fractional amount (between 0 and 1) of some set to be added into a solution. When the fractional solution is constructed, it is rounded to a feasible solution using pipage rounding. The pipage rounding technique rounds the fractional solutions to a discrete one using randomized directions. This algorithm achieves an approximation ratio of $(1 - \frac{1}{e})$ for the monotone submodular function maximization subject to a matroid constraint. This approximation ratio is optimal under the same conditions stated by Feige (1998).

Filmus and Ward (2012) investigated local search algorithms, instead of greedy-like algorithms, to solve the maximum coverage problem. Their paper considered the maximum coverage problem subjected to a matroid constraint (MCPM). They used a swap local search algorithm with an auxiliary objective function to solve the problem and proved that the approximation ratio is $(1 - \frac{1}{e})$. This auxiliary objective function gives more weight to elements covered multiple times. The achieved approximation ratio is optimal under the conditions of Feige (1998).

Another technique that will be investigated in this thesis, is the Lagrange relaxation method combined with subgradient optimization to solve integer programs. This method was first used in Held and Karp (1970) and Held and Karp (1971). In these two papers, the Lagrange relaxation method with subgradient optimization was used to solve the symmetric traveling salesman problem. In essence, the Lagrange relaxation method relaxes an integer program by removing one or several constraints and adding these in the objective function, multiplied by penalty vector (the Lagrange multiplier vector). These multipliers are chosen such that the objective function is penalized if a constraint is violated. The Lagrange relaxation method is particularly useful if the relaxed program is computationally easier to solve than the original program. If the problem is a maximization (minimization) problem, solving the relaxed problem gives upper (lower) bounds for every Lagrange multiplier vector. Subgradient optimization then tries to find the multiplier vector that minimizes (maximizes) the upper (lower) bounds of the problem. It does so by updating the multiplier vector based on the value of the relaxed constraints. The penalty increases if the relaxed constraints are violated, and decreases if the relaxed constraints are not violated. Held, Wolfe, and Crowder (1974) proved the convergence of the subgradient method if the chosen step sizes satisfy certain criteria.

The final technique we will use is tabu search. Tabu search is a metaheuristic that is used quite frequently to enhance local search algorithms, as described in Glover (1989). The simplest variant of tabu search is the so-called short-term memory tabu search. The short-term memory tabu search introduces a tabu list during a local search. This list stores a fixed number of solutions previously visited by the local search algorithm. The list uses a first-in-first-out structure, where the first added solution is removed if the list is full. Whenever a solution is in the tabu list, the local search cannot select that solution anymore, essentially forcing the local search to consider other solutions. Because of the tabu list, the local search can escape local optima and potentially find better solutions.

Given this large number of algorithms that exist to solve the maximum coverage problem and its variants, it makes sense to select a few existing algorithms and investigate their performance. However, first we will formally define the maximum coverage problem and the two variants considered in this thesis formally.

3 The Maximum Coverage Problem

This part of the thesis introduces the variants of the maximum coverage problem (MCP) that will be investigated in this thesis. Section 3.1 gives the notational convention used in this thesis. Section 3.2 defines the maximum coverage problem formally. An integer program for the problem is set up in Section 3.3. Finally, Section 3.4 defines the considered variants of the MCP of this thesis.

3.1 Notational Convention

As this thesis deals with sets of elements, sets of sets and sets of sets of sets, we use the following notational convention to differentiate between the different mathematical objects:

1. Mathematical objects that are not sets (such as elements, real-valued functions or indices) are denoted in lowercase italic font, for instance u or w.
2. Sets of elements are denoted with uppercase italic, for instance $U = \{x, y, z\}$.
3. Sets of sets are denoted with a calligraphic font, for instance $\mathcal{F} = \{S_1, \ldots, S_F\}$.
4. Sets of sets of sets are denoted with boldface, for instance $\mathbf{F} = \{\mathcal{Y} \subseteq \mathcal{F} : |\mathcal{Y}| \leq n\}$.

The only exception to this rule are cardinalities of sets, whose symbol and meaning are specified explicitly. We will use this convention throughout this thesis.

3.2 Description of the Problem

For the MCP we need a set of elements U, a family of subsets $\mathcal{F} = \{S_1, \ldots, S_F\}$ with $S_f \subseteq U$ for $f \in \{1, \ldots, F\}$ and a weight function $w : U \to \mathbb{R}_{\geq 0}$. For convenience, we extend the definition of w for a set $S \subseteq U$ and a set of sets $\mathcal{Y} \subseteq \mathcal{F}$ as

$$w(S) = \sum_{u \in S} w(u)$$

and

$$w(\mathcal{Y}) = w\Big(\bigcup_{S \in \mathcal{Y}} S\Big).$$

As explained in the introduction, the objective of the MCP is to select sets from $\mathcal{F}$ such that the weight of the covered elements in these sets is maximal. To be able to describe the variants of the MCP in general, we introduce a set of feasible solutions $\mathbf{F} \subseteq 2^{\mathcal{F}}$ and restrict solutions, subsets of $\mathcal{F}$, to lie in the set $\mathbf{F}$. Then, we can define a general maximum coverage problem as follows:

$$\max_{\mathcal{Y} \in \mathbf{F}} w(\mathcal{Y}).$$

By making specific choices for $\mathbf{F}$, we can obtain different variants of the MCP. In Section 3.4 we will define $\mathcal{F}$ for the three variants considered in this thesis: the standard maximum coverage problem (MCP), the maximum coverage problem over a matroid (MCPM) and the maximum coverage problem with budget constraints (MCPBC). But first we will introduce an integer program for the general maximum coverage problem.

3.3 Integer Program Formulation

In order to assess the heuristic methods, we determine optimal solutions using an integer program (IP). The parameter a_{uf} is one if and only if element $u \in U$ is contained in S_f, $f \in \{1, \ldots, F\}$. The variable x_f is one if and only if we choose set S_f in our solution, $f \in \{1, \ldots, F\}$. The variable y_u is one if and only if element $u \in U$ is covered. We also introduce the nonnegative parameters β_{fi} and n_i for all $f \in \{1, \ldots, F\}$

and $i \in \{1, \ldots, m\}$. These parameters will be used to restrict our solutions to the set $\mathbf{F}$ using m linear constraints.

$$\max \sum_{u \in U} w(u) y_u \tag{1}$$

$$\text{s.t. } y_u \leq \sum_{f=1}^{F} a_{uf} x_f \qquad \forall\, u \in U, \tag{2}$$

$$\sum_{f=1}^{F} \beta_{fi} x_f \leq n_i \qquad \forall i \in \{1, \ldots, m\}, \tag{3}$$

$$y_u \in \{0, 1\} \qquad \forall\, u \in U, \tag{4}$$

$$x_f \in \{0, 1\} \qquad \forall\, f \in \{1, \ldots, F\}. \tag{5}$$

In the model (1)-(5), we maximize the total weight of the covered elements in (1). Constraint (2) ensures that if we do no choose any set that contains element $u \in U$, then we do not cover that particular element. Constraint (3) states that the chosen solution should be feasible. By choosing specific values for the parameters β_{fi}, n_i and m, many different feasible sets $\mathbf{F}$ can be modeled. Finally, constraints (4) and (5) are the domain constraints of the variables x_f and y_u respectively.

3.4 Variants of the Maximum Coverage Problem

In this thesis, we evaluate the heuristics for three variants of the maximum coverage problem, each with their own choice of $\mathbf{F}$. These variants are the standard maximum coverage problem (MCP), the maximum coverage problem over a matroid (MCPM) and the maximum coverage problem with budget constraints (MCPBC).

Maximum Coverage Problem

The MCP restricts the maximum cardinality of a solution. That is, the feasible set is

$$\mathbf{F} = \{\mathcal{Y} \subseteq \mathcal{F} : |\mathcal{Y}| \leq n\}$$

for some positive integer n. To implement this choice of $\mathbf{F}$ in the IP of Section 3.3, we set $m = 1$, $n_1 = n$ and $\beta_{f1} = 1$ for all $f \in \{1, \ldots, F\}$. Constraint (3) would then become

$$\sum_{f=1}^{F} x_f \leq n. \tag{MCP(3)}$$

Maximum Coverage Problem over a Matroid

For the MCPM, our set of feasible solutions $\mathbf{F}$ is equal to the collection of independent sets of a matroid $\mathfrak{m}$. Formally, a matroid $\mathfrak{m}$ is defined as a tuple $(\mathcal{G}, \mathbf{I})$. $\mathcal{G}$ is the ground set and $\mathbf{I} \subseteq 2^{\mathcal{G}}$ is the collection of independent sets. $\mathfrak{m}$ is a matroid if and only if

1. $\mathbf{I}$ is not empty,
2. if $\mathcal{A} \in \mathbf{I}$ and $\mathcal{B} \subset \mathcal{A}$, then $\mathcal{B} \in \mathbf{I}$,
3. for all $\mathcal{A}, \mathcal{B} \in \mathbf{I}$ with $|\mathcal{B}| < |\mathcal{A}|$ it holds that there exists an $X \in \mathcal{A} \setminus \mathcal{B}$ such that $\mathcal{B} \cup \{X\} \in \mathbf{I}$.

The rank of the matroid $\mathfrak{m}$ is defined as the maximum cardinality of the sets in $\mathbf{I}$. In our setting, the ground set $\mathcal{G}$ is equal to $\mathcal{F}$ and the independent sets $\mathbf{I}$ form our set of feasible solutions $\mathbf{F}$.

However, describing an arbitrary matroid with linear constraints proves difficult, if not impossible. Therefore, we restrict $\mathfrak{m}$ to be a partition matroid whenever we need a particular implementation of the MCPM:

the maximum coverage problem over a partition matroid (MCPPM). To define a partition matroid, the set $\mathcal{F}$ is partitioned into m disjoint sets $\mathcal{F}_1, \ldots, \mathcal{F}_m$ and choose nonnegative integers $n_1, \ldots, n_m$. A solution is feasible if and only if it contains at most n_i sets from $\mathcal{F}_i$. In other words,

$$\mathbf{I} = \mathbf{F} = \big\{\mathcal{Y} \subseteq \mathcal{F} : |\mathcal{Y} \cap \mathcal{F}_i| \leq n_i, \forall\, i \in \{1, \ldots, m\}\big\}.$$

The rank of this matroid is defined as $n = \sum_{i=1}^{m} n_i$. We can describe this set using m linear constraints. We define the binary parameter b_{fi} to be one if and only if set S_f is in partition $\mathcal{F}_i$, $f \in \{1, \ldots, F\}$ and $i \in \{1, \ldots, m\}$. To implement this into our model, we set $\beta_{fi} = b_{fi}$. We can then write the feasibility constraint (3) as

$$\sum_{f=1}^{F} b_{fi} x_f \leq n_i \qquad \forall\, i \in \{1, \ldots, m\}. \qquad \text{MCPPM(3)}$$

Note that the approximation ratios of the algorithms, described in the next section, are still based on a general matroid constraint. Whenever we talk about theoretical properties, such as approximation ratios, we mean a general matroid constraint. However, as our only implementation is a partition matroid, numerical results are based on the MCPPM.

Maximum Coverage Problem with Budget Constraints

For the MCPBC, we introduce a cost function $c : \mathcal{F} \to \mathbb{R}_{>0}$. For convenience, we extend the definition of c to a subset $\mathcal{Y} \subseteq \mathcal{F}$ as

$$c(\mathcal{Y}) = \sum_{S \in \mathcal{Y}} c(S).$$

A solution in the MCPBC is feasible if and only if the costs of that solution are less than a predefined budget n. That is, $\mathbf{F} = \{\mathcal{Y} \subseteq \mathcal{F} : c(\mathcal{Y}) \leq n\}$. To describe this set in the IP, we set $m = 1$, $n_1 = n$ and $\beta_{f1} = c(S_f)$ for all $f \in \{1, \ldots, F\}$. We can then model the feasibility constraint (3) as

$$\sum_{f=1}^{F} c(S_f) x_f \leq n. \qquad \text{MCPBC(3)}$$

In Lemmas 1-4 in Appendix A we give proofs that the MCP is a special case of both the MCPM and the MCPBC. Furthermore, we prove that neither MCPM is a special case of MCPBC nor is MCPBC a special case of MCPM.

4 Solution Methods

In this section, we describe different solution methods for the maximum coverage problem and its variants. Section 4.1 defines the aforementioned greedy algorithm and its implementation. Sections 4.2 and 4.3 discusses the oblivious and non-oblivious swap local search, as mentioned in Filmus and Ward (2012). Section 4.4 introduces a Lagrange relaxation-based heuristic to solve the MCPBC. Finally, Section 4.5 considers a tabu swap local search and combines it with a relaxation for the MCPBC.

4.1 Greedy Algorithm

The greedy algorithm (Greedy) constructs a feasible solution for the MCP and its variants by iteratively choosing sets $S \in \mathcal{F}$ that maximize a certain criteria function until no more sets can be added. Let us denote the criteria function with $g : \mathcal{F} \times \mathbf{F} \to \mathbb{R}$, which assigns a real value to a set $S \in \mathcal{F}$ and a current solution $\mathcal{Y} \in \mathbf{F}$. The criteria function depends on the considered variant of the MCP. Both the MCP and the MCPM use

$$g(S, \mathcal{Y}) = w(\mathcal{Y} \cup \{S\}) - w(\mathcal{Y}),$$

such that the greedy algorithm adds the set that causes the largest marginal weight increase. On the other hand, the MCPBC uses

$$g(S, \mathcal{Y}) = \frac{w(\mathcal{Y} \cup \{S\}) - w(\mathcal{Y})}{c(S)}$$

(Khuller et al., 1999), so that it prefers sets with the highest marginal weight increase-to-cost ratio.

The approximation ratio of the greedy algorithm differs for the considered variant of the MCP. For the MCP itself, it is $(1 - \frac{1}{e})$, which is the highest possible unless P=NP (Feige, 1998). The greedy algorithm on the MCPM unfortunately has only an approximation ratio of $\frac{1}{2}$ (Filmus & Ward, 2012), and on the MCPBC it does not have any approximation ratio (Khuller et al., 1999).

As not having an approximation ratio can be undesirable, Khuller et al. (1999) proposed one simple adjustment to the greedy algorithm so that the approximation ratio becomes $\frac{1}{2}(1 - \frac{1}{e})$. After constructing a greedy solution, we check whether there is a solution $\mathcal{X}$ with $|\mathcal{X}| = 1$ such that $w(\mathcal{X}) > w(\mathcal{Y})$. If so, we return $\mathcal{X}$ instead of $\mathcal{Y}$. We will use this modified version for the MCPBC, which we also call Greedy. The pseudo code of Greedy is given in Algorithm 1.

Algorithm 1: Greedy.

```
1  Y ← ∅;
2  R ← F;
3  while R ≠ ∅ do
4  |  S ← arg max_{S∈R} g(S, Y);
5  |  if Y ∪ {S} ∈ F then
6  |  |  Y ← Y ∪ {S};
7  |  end
8  |  R ← R \ {S};
9  end
10 if this problem is a MCPBC then
11 |  X ← arg max_{X∈F: |X|=1} w(X);
12 |  if w(X) > w(Y) then
13 |  |  Y ← X;
14 |  end
15 end
16 return Y;
```

Lines 1-2 of Algorithm 1 initialize the solution $\mathcal{Y}$ and the remaining sets to be considered $\mathcal{R}$. Lines 3-9 iteratively add sets S to the solution that maximize $g(S, \mathcal{Y})$. If the resulting solution $\mathcal{Y} \cup \{S\}$ is not feasible, S will not be added to the solution. Lines 10-15 describe the modification proposed by Khuller et al. (1999) to give Greedy an approximation ratio for the MCPBC.

To improve the approximation ratio of the greedy algorithm for the MCPBC, Khuller et al. (1999) introduced another adjustment to the greedy algorithm. In this variant, which uses partial enumeration, we choose an integer κ and enumerate all solutions $\mathcal{X}$ with cardinality strictly less than κ. We call the solution with the greatest weight $\mathcal{H}_1$. Then, we perform the greedy algorithm on every solution $\mathcal{X}$ with cardinality κ. We call the greedily extended solution with the greatest weight $\mathcal{H}_2$. We return either $\mathcal{H}_1$ or $\mathcal{H}_2$, whichever has higher weight. This variant, which we will call CostGreedy does achieve an approximation ratio of $(1 - \frac{1}{e})$ for $\kappa \geq 3$. Khuller et al. (1999) proved that $(1 - \frac{1}{e})$ is the highest approximation ratio possible for the MCPBC under similar conditions as in Feige (1998). The pseudo code is given in Algorithm 2.

Algorithm 2: CostGreedy.

$\mathcal{H}_1 \leftarrow \underset{\mathcal{X}\in\mathbf{F}:|\mathcal{X}|<\kappa}{\arg\max}\ w(\mathcal{X})$;
$\mathcal{H}_2 \leftarrow \varnothing$;
foreach $\mathcal{X} \in \mathbf{F} : |\mathcal{X}| = \kappa$ **do**
  $\mathcal{Y} \leftarrow$ result of Greedy without lines 10-15 on $\mathcal{X}$;
  if $w(\mathcal{Y}) > w(\mathcal{H}_2)$ **then**
    $\mathcal{H}_2 \leftarrow \mathcal{Y}$;
  end
end
if $w(\mathcal{H}_1) \geq w(\mathcal{H}_2)$ **then**
  return $\mathcal{H}_1$;
else
  return $\mathcal{H}_2$;
end

4.2 Oblivious Swap Local Search

The oblivious swap local search (OblSwap) tries to improve the objective value of an initial solution by iteratively swapping several sets from the current solution with sets not in the solution. To accomplish this, the algorithm inspects the objective values of solutions that differ at most k sets from the current solution $\mathcal{Y}$. If we denote this neighborhood with $\mathbf{N}_k(\mathcal{Y})$, we can define it as:

$$\mathbf{N}_k(\mathcal{Y}) = \{\mathcal{X} \in \mathbf{F} : |\mathcal{X} \setminus \mathcal{Y}| \leq k, |\mathcal{Y} \setminus \mathcal{X}| \leq k\}.$$

To be more exact, OblSwap picks iteratively the solution from the neighborhood $\mathbf{N}_k(\mathcal{Y})$ with the highest weight. It keeps doing so until the objective value cannot be increased further. The pseudo code of the oblivious swap local search is given in Algorithm 3.

Algorithm 3: OblSwap.

$\mathcal{Y} \leftarrow$ some feasible solution;
repeat
  $\mathcal{Y}_{old} \leftarrow \mathcal{Y}$;
  $\mathcal{Y} \leftarrow \underset{\mathcal{X}\in\mathbf{N}_k(\mathcal{Y})}{\arg\max}\ w(\mathcal{X})$;
until $\mathcal{Y}_{old} = \mathcal{Y}$;
return $\mathcal{Y}$;

In this thesis, we use the outcome of Greedy as the initial solution for OblSwap. As this local search heuristic is guaranteed to not decrease the weight of the solution, this algorithm has an approximation ratio of the greedy algorithm or higher. For the MCP, the approximation ratio is $(1 - \frac{1}{e})$, which is again the highest one possible (unless P=NP). If we use OblSwap on the MCPM, the approximation ratio is $\frac{n-1}{2n-k-1}$ (Filmus & Ward, 2012). Little research has been done on the performance of the oblivious swap local search algorithm on the MCPBC, but if we use Greedy from the previous section, the approximation ratio is at least $\frac{1}{2}(1 - \frac{1}{e})$ and at most $\frac{1}{2}$ (see Lemma 5 in Appendix A). This upper bound of $\frac{1}{2}$ holds only for $k = 1$.

4.3 Non-Oblivious Swap Local Search

The non-oblivious swap local search (NonOblSwap) was first introduced in Filmus and Ward (2012). This heuristic chooses the solution from the neighborhood $\mathbf{N}_k(\mathcal{Y})$ (defined in Section 4.2) that maximizes a modified weight function $w' : \mathbf{F} \to \mathbb{R}$. This function w' assigns more weight to elements that are covered multiple times. Formally, the function is defined as

$$w'(\mathcal{Y}) = \sum_{u\in U} \alpha_{h_u(\mathcal{Y})} w(u)$$

with $h_u(\mathcal{Y}) = |\{S \in \mathcal{Y} : u \in S\}|$ the number of sets in $\mathcal{Y}$ that contain the element u and α_i some constants for $i \in \{0, 1, 2, \ldots\}$. Note that if we choose $\alpha_0 = 0$ and $\alpha_i = 1$ for all $i \in \{1, 2, \ldots\}$, then w' equals the original weight function w. Filmus and Ward (2012) showed that the following choices for α_i are optimal for the MCP and MCPM:

$$\alpha_0 = 0, \tag{6}$$

$$\alpha_1 = 1 - \frac{1}{e^{[n]}}, \tag{7}$$

$$\alpha_{i+1} = (i+1)\alpha_i - i\alpha_{i-1} - \frac{1}{e^{[n]}}, \tag{8}$$

where $e^{[n]} = \sum_{l=0}^{n-1} \frac{1}{l!} + \frac{1}{(n-1)!(n-1)}$. The pseudo code of this heuristic is given in Algorithm 4.

Algorithm 4: NonOblSwap.

```
1 𝒴 ← some feasible solution;
2 repeat
3 |   𝒴_old ← 𝒴;
4 |   𝒴 ← arg max_{𝒳∈N_k(𝒴)} w'(𝒳);
5 until 𝒴_old = 𝒴;
6 return 𝒴;
```

Just as with OblSwap, we use the outcome of Greedy as an initial solution. Filmus and Ward (2012) proved that this algorithm achieves an approximation ratio of $(1 - \frac{1}{e})$ for the MCP and MCPM. For the MCPBC, however, the approximation ratio is unknown. The initial greedy solution does not give an approximation ratio, as increasing w' can lead to decreasing w. We show in Lemma 5 in Appendix A that the approximation ratio is at most $\frac{1}{2}$ for the MCPBC whenever $k = 1$.

4.4 The Lagrange Multiplier Swap Local Search

Filmus and Ward (2012) designed the NonOblSwap heuristic to achieve an approximation ratio of $(1 - \frac{1}{e})$ on the MCPM and MCP. With the MCPM and MCP, as the weights of each element is positive, maximal weight is always achieved for solutions with cardinality n. Therefore, the heuristic can use 1-exchanges in order to improve on the weight w or modified function w'. However, with the MCPBC, maximum weight can occur at any cardinality of the solution due to the budget constraint. Thus the algorithm can get stuck sooner in suboptimal local optima considering only feasible 1-exchanges.

To solve this problem, we consider a relaxation of the budget constraint. That is, we temporarily allow the heuristic to consider infeasible solutions so that we can escape local optima. One way to do this is using Lagrange multipliers. That is, we relax the constraint $c(\mathcal{Y}) \leq n$ by adding it into the objective function with a penalty factor. Thus, the new objective function becomes

$$w^\lambda(\mathcal{Y}) = w(\mathcal{Y}) + \lambda(n - c(\mathcal{Y}))$$

for some scalar multiplier $\lambda \geq 0$. Furthermore, to not leave solutions unconstrained, we add the cardinality constraint $|\mathcal{Y}| \leq M$ to the problem, where $M = \max_{\mathcal{Y} \in \mathbf{F}} |\mathcal{Y}|$. The constraint $|\mathcal{Y}| \leq M$ is implied by $c(\mathcal{Y}) \leq n$ with this specific choice of M. We denote the relaxed set of feasible solutions with

$$\mathbf{F}_R = \{\mathcal{Y} \subseteq \mathcal{F} : |\mathcal{Y}| \leq M\}.$$

Normally, with Lagrange relaxations for an integer program, one could use the subgradient method to obtain upper and lower bounds for the problem (Held et al., 1974). That is, the problem is solved to optimality for some $\lambda \geq 0$, with the adjusted objective function w^λ as upper bound for the optimal original objective function w. Then a feasible solution is constructed from the (usually infeasible) obtained solution to get a lower bound on the optimal objective. After that, λ is updated according to some rule and the process is

repeated until a certain termination criteria is met.

In our case, even when relaxing $c(\mathcal{Y}) \leq n$, we are not able to solve the problem to optimality in polynomial time. Using an IP to maximize w^λ is also not desired, as then we could also solve the MCPBC itself with an IP. However, we can use the inexact methods presented in the previous sections. One such method we could use is the swap local search. The swap local search will use the relaxed neighborhood $\mathbf{N}_k^R(\mathcal{Y})$, defined as

$$\mathbf{N}_k^R(\mathcal{Y}) = \{\mathcal{X} \in \mathbf{F}_R : |\mathcal{X} \setminus \mathcal{Y}| \leq k, |\mathcal{Y} \setminus \mathcal{X}| \leq k\}.$$

To construct feasible solutions out of infeasible ones we use a method based on the greedy algorithm combined with a swap local search. That is, we iteratively remove the set S from the infeasible solution $\mathcal{Y}$ with the highest cost-to-weight decrease ratio until the solution is feasible. However, if there exists a set in the solution whose removal does not affect the weight of the solution, we remove that set. We do this extra step so that we prevent division by zero. The pseudo code for this method (INFEAS2FEAS) is given in Algorithm 5.

Algorithm 5: INFEAS2FEAS

$\mathcal{Y} \leftarrow$ some (infeasible) solution;
while $c(\mathcal{Y}) > n$ **do**
  if *there exists a* $S \in \mathcal{Y}$ *such that* $w(\mathcal{Y}) = w(\mathcal{Y} \setminus \{S\})$ **then**
    $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{S\}$;
  else
    $S \leftarrow \arg\max_{S \in \mathcal{Y}} \frac{c(S)}{w(\mathcal{Y}) - w(\mathcal{Y} \setminus \{S\})}$;
    $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{S\}$;
  end
end
$\mathcal{Y} \leftarrow$ result of OBLSWAP on $\mathcal{Y}$;
return $\mathcal{Y}$;

After a feasible solution is constructed from the possibly infeasible one, update our multiplier λ. The update for λ is based on the recommendation in Held et al. (1974), which is for the ith iteration

$$\lambda_{i+1} = \lambda_i + s_i \rho_i \frac{w^{\lambda_i}(\mathcal{Y}_i) - LB_i}{||s_i||^2},$$

with $\mathcal{Y}_i$ the solution that maximizes w^{λ_i}, $s_i = c(\mathcal{Y}_i) - n$ the subgradient, LB_i a lower bound on the maximal feasible weight $\max_{\mathcal{Y} \in \mathbf{F}} w(\mathcal{Y})$, ρ_i a positive decreasing sequence with $\lim_{i \to \infty} \rho_i = 0$ and $||\cdot||$ the Euclidean norm on the domain of s_i. As s_i is a scalar in our context, the Euclidean norm equals the absolute value. The idea is that λ_{i+1} will be increased if $s_i > 0$, which is whenever $\mathcal{Y}_i$ is infeasible. By increasing λ_{i+1} we then penalize infeasible costs even higher, thus forcing the costs of the next solution to lie lower. Similarly, if $s_i < 0$, the solution $\mathcal{Y}_i$ is feasible and we decrease λ_{i+1} such that the costs of the next solution will be higher. As LB_i should form a lower bound on the maximal feasible weight, we can set LB_i to be the maximum weight of all encountered feasible solutions so far.

However, the update for λ_{i+1} suggested by Held et al. (1974) is only valid whenever $w^{\lambda_i}(\mathcal{Y}_i) \geq LB_i$. As our maximization of w^{λ_i} is inexact, we need to make an adjustment to guarantee this inequality. Every iteration we use $\mathcal{Y}_{i-1}^*$, the feasible solution with the highest weight so far, as the initial solution for the swap local search to maximize w^{λ_i}. As $LB_{i-1} = w(\mathcal{Y}_{i-1}^*) \leq w^{\lambda_i}(\mathcal{Y}_{i-1}^*)$ and the swap local search never decreases w^{λ_i}, our desired inequality is met if $\mathcal{Y}_{i-1}^*$ is not updated that iteration (thus if $LB_{i-1} = LB_i$). If $\mathcal{Y}_{i-1}^*$ is updated and $LB_i = w(\mathcal{Y}_i^*) > w^{\lambda_i}(\mathcal{Y}_i)$, we can use the previous LB_{i-1} for the update of λ_i.

We terminate the algorithm after a fixed number of iterations N_L. We also terminate the algorithm whenever we encounter a solution with costs equal to n. The full pseudo code for this Lagrange Multiplier method (LAGRAN) is given in Algorithm 6.

Algorithm 6: LAGRAN.

1 $\mathcal{Y}_0^* \leftarrow$ some feasible solution;
2 $\lambda_1 \leftarrow 0$;
3 $LB_0 \leftarrow w(\mathcal{Y}_0^*)$;
4 **for** $i \leftarrow 1$ ***to*** N_L **do**
5 $\quad \mathcal{Y}_i \leftarrow \mathcal{Y}_{i-1}^*$;
6 $\quad$ //Swap local search;
7 $\quad$ **repeat**
8 $\quad\quad \mathcal{Y}_{old} \leftarrow \mathcal{Y}_i$;
9 $\quad\quad \mathcal{Y}_i \leftarrow \underset{\mathcal{X} \in \mathbf{N}_k^R(\mathcal{Y})}{\arg\max}\ w^{\lambda_i}(\mathcal{X})$;
10 $\quad$ **until** $\mathcal{Y}_{old} = \mathcal{Y}_i$;
11 $\quad$ //Make solution feasible;
12 $\quad \mathcal{Y}_{feas,i} \leftarrow$ result of INFEAS2FEAS on $\mathcal{Y}_i$;
13 $\quad$ //Update bounds;
14 $\quad$ **if** $w(\mathcal{Y}_{feas,i}) > LB_{i-1}$ **then**
15 $\quad\quad \mathcal{Y}_i^* \leftarrow \mathcal{Y}_{feas,i}$;
16 $\quad\quad LB_i \leftarrow w(\mathcal{Y}_{feas,i})$;
17 $\quad$ **else**
18 $\quad\quad \mathcal{Y}_i^* \leftarrow \mathcal{Y}_{i-1}^*$;
19 $\quad\quad LB_i \leftarrow LB_{i-1}$;
20 $\quad$ **end**
21 $\quad$ //Check for termination;
22 $\quad$ **if** $c(\mathcal{Y}_{feas,i}) = n$ **then**
23 $\quad\quad \mathcal{Y}_{N_L}^* \leftarrow \mathcal{Y}_i^*$;
24 $\quad\quad LB_{N_L} \leftarrow LB_i$;
25 $\quad\quad$ **break**;
26 $\quad$ **end**
27 $\quad$ //Update Lagrange multiplier;
28 $\quad s_i \leftarrow (c(\mathcal{Y}_i) - n)$;
29 $\quad$ **if** $w^{\lambda_i}(\mathcal{Y}_i) \geq LB_i$ **then**
30 $\quad\quad \lambda_{i+1} \leftarrow \lambda_i + s_i \frac{1}{i} \frac{w^{\lambda_i}(\mathcal{Y}_i) - LB_i}{s_i^2}$;
31 $\quad$ **else**
32 $\quad\quad \lambda_{i+1} \leftarrow \lambda_i + s_i \frac{1}{i} \frac{w^{\lambda_i}(\mathcal{Y}_i) - LB_{i-1}}{s_i^2}$;
33 $\quad$ **end**
34 **end**
35 **return** $\mathcal{Y}_{N_L}^*$;

The update for the multiplier λ in lines 30 and 32 is written such that the structure of the update as in Held et al. (1974) is maintained. For the constants ρ_i we chose $\frac{1}{i}$. Note that this algorithm will not produce any upper bounds on the optimal objective value $\max_{\mathcal{Y} \in \mathbf{F}} w(\mathcal{Y})$, as the maximization of $w^{\lambda}(\mathcal{Y})$ is inexact. Our initial feasible solution $\mathcal{Y}_0^*$ is the outcome of GREEDY.

Furthermore, $w^{\lambda}(\mathcal{Y})$ is a non-monotone submodular function, which means a local search or greedy algorithm does not have a performance guarantee. Suppose we replace the local search (lines 7-11) by an α-approximation algorithm for maximizing $w^{\lambda_i}(\mathcal{Y})$. In that case, upper bounds could be obtained by setting $UB = \frac{1}{\alpha} l^{\lambda_i}(\mathcal{X})$, where $\mathcal{X}$ is the result of the α-approximation algorithm. The proof for this fact is given in Lemma 6 of Appendix A.

4.5 Tabu Swap Local Search

Another method to aid the swap local search in escaping local optima is the principle of tabu search (Tabu). With tabu search, one keeps a list $\mathbf{L}$ of the L last visited solutions, and restricts the local search to only consider solutions not in the list (Glover, 1989). That is, the tabu list $\mathbf{L}$ can be viewed as an ordered collection of solutions with a maximum size L. Whenever the swap local search updates its current solution, the solution is also added to the tabu list. As long as a solution is in the tabu list, the swap local search will ignore this solution. This tabu list will thus force the swap local search to keep updating the current solution, even if the solution is a local optimum. The tabu list discards the oldest entry if another solution is added and L solutions are already present in the list.

Furthermore, to aid the local search for the MCPBC, we could temporarily allow for infeasible solutions. We allow the local search to ignore the constraint $c(\mathcal{Y}) \leq n$ for N_I iterations before forcing to local search back to feasibility. That is, we use the relaxed neighborhood $\mathbf{N}_k^R(\mathcal{Y})$ for the local search whenever at least one of the last N_I iterations was feasible. The regular neighborhood $\mathbf{N}_k(\mathcal{Y})$ is used if all of the last N_I iterations were infeasible. It could happen that the regular neighborhood is empty, especially if a large number of infeasible iterations have occurred. In this case, we use Infeas2Feas (see Section 4.4) to force the solution into feasibility.

We terminate the algorithm if the feasible solution with the highest weight has not updated for N_T iterations. These ideas are summarized in Algorithm 7.

Algorithm 7: Tabu.

$\mathcal{Y} \leftarrow$ some feasible solution;
$\mathcal{Y}^* \leftarrow \mathcal{Y}$;
$w^* \leftarrow w(\mathcal{Y}^*)$;
$i \leftarrow 0$;
repeat
  if $i < N_I$ **then**
    $\mathcal{Y} \leftarrow \underset{\mathcal{X} \in \mathbf{N}_k^R(\mathcal{Y}) \setminus \mathbf{L}}{\arg\max} \; w(\mathcal{X})$;
  else
    if $(\mathbf{N}_1(\mathcal{Y}) \setminus \mathbf{L}) \neq \varnothing$ **then**
      $\mathcal{Y} \leftarrow \underset{\mathcal{X} \in \mathbf{N}_k(\mathcal{Y}) \setminus \mathbf{L}}{\arg\max} \; w(\mathcal{X})$;
    else
      $\mathcal{Y} \leftarrow$ result of Infeas2Feas on $\mathcal{Y}$;
    end
  end
  Add $\mathcal{Y}$ to $\mathbf{L}$;
  If already L solutions in $\mathbf{L}$, remove the oldest entry;
  if $c(\mathcal{Y}) \leq n$ **then**
    if $w(\mathcal{Y}) \geq w^*$ **then**
      $\mathcal{Y}^* \leftarrow \mathcal{Y}$;
      $w^* \leftarrow w(\mathcal{Y})$;
    end
    $i \leftarrow 0$;
  else
    $i \leftarrow i + 1$;
  end
until $\mathcal{Y}^*$ *has not updated for* N_T *iterations*;

Our initial solution $\mathcal{Y}$ is the outcome of Greedy. As we show in Appendix B that in some instances of the MCPBC choosing $N_I = 0$ yields higher results.

A possible reason as to why not allowing infeasible solution ($N_I = 0$) sometimes yields better results than allowing for infeasible solutions ($N_I > 0$) is that we did not penalize the potential infeasibility in any way. Therefore, we will only penalize infeasible solutions with the following modified weight function defined as

$$w_+^\lambda(\mathcal{Y}) = w(\mathcal{Y}) - \lambda(\max\{c(\mathcal{Y}) - n, 0\})$$

with $\lambda \geq 0$. This function $w_+^\lambda(\mathcal{Y})$ equals the weight function $w(\mathcal{Y})$ whenever the solution $\mathcal{Y}$ is feasible. If $\mathcal{Y}$ is not feasible, the function $w_+^\lambda(\mathcal{Y})$ is smaller than, or equal to, the weight of that solution. This way, we still allow the algorithm to consider infeasible solutions, but force it to stay close to feasibility, as we penalize for costs greater than the budget n. Keep in mind that the algorithm TABU does not use this penalty function, which is the same as setting $\lambda = 0$.

So, we can use the modified weight function $w_+^\lambda(\mathcal{Y})$ in line 7 of Algorithm 7. For the change to have any effect, we should set $N_I > 0$. Furthermore, the found solution could heavily depend on the choice of λ. As the penalty of $w_+^\lambda(\mathcal{Y})$ is linear, just as a Lagrange multiplier, one could use the last λ from LAGRAN (see Section 4.4) as the value of λ for this tabu swap local search. We shall call this variant TABULAGRAN, as it uses LAGRAN as its input.

Another choice for λ that does not depend on a different method and seems to perform well in practice is choosing $\lambda = \lambda_R = \frac{w(\mathcal{Y})}{c(\mathcal{Y})}$. Note that this multiplier does depend on the currently considered solution $\mathcal{Y}$. If we rewrite the modified weight function $w_+^\lambda(\mathcal{Y})$ with this particular choice of $\lambda = \lambda_R$, we get

$$w_+^{\lambda_R}(\mathcal{Y}) = \begin{cases} w(\mathcal{Y})\dfrac{n}{c(\mathcal{Y})} & \text{if } c(\mathcal{Y}) > n \\ w(\mathcal{Y}) & \text{if } c(\mathcal{Y}) \leq n \end{cases}.$$

So, this choice of $\lambda = \lambda_R$ essentially penalizes with a multiplicative factor, instead of an additive one. This multiplicative penalizing factor $\frac{n}{c(\mathcal{Y})}$ is easy to interpret. For instance, we have an infeasible solution $\mathcal{Y}$ whose costs are $x\%$ larger than the budget n. Then, if we want the tabu local search to prefer $\mathcal{Y}$ over some feasible solution $\mathcal{X}$, then the weight of $\mathcal{Y}$ has to be more than $x\%$ larger than the weight of $\mathcal{X}$. In other words, any relative increase of the costs over the budget must be met with an equal or higher relative increase of the weight. We shall call this method TABURATIO, as the choice of $\lambda = \lambda_R$ is the ratio of the weight over the costs of the solution.

5 Numerical Results

This section presents the numerical results of this thesis. Section 5.1 describes the algorithm used to generate random problem instances for the MCP, MCPPM and MCPBC. Sections 5.2, 5.3 and 5.4 give the numerical results for the MCP, MCPPM and MCPBC respectively. These sections will compare the different methods and explain the difference in performance between them.

5.1 Random Instance Generation

First, we describe how problem instances for the maximum coverage problem and its variants are generated. We base our methods largely on Resende (1998), who generated maximum coverage problems with this method. In this method, an element $u \in U$ is interpreted as a demand location on the two dimensional unit square with Cartesian coordinates (x_u, y_u). Likewise, a set $S_f \in \mathcal{F}$ is a facility in the same space with coordinates (x_f, y_f). An element $u \in U$ is contained in a set S_f if and only if the Euclidean distance $d_{uf} = \sqrt{(x_u - x_f)^2 + (y_u - y_f)^2}$ is smaller than r_{max}, the cover radius of the facility. Weights of elements are uniformly distributed between a predefined minimum w_{min} and maximum w_{max}. The pseudo code for this generation method is given in Algorithm 8.

Algorithm 8: Random Maximum Coverage Instance Generator.

1 **for** $u \leftarrow 1$ ***to*** $|U|$ **do**
2 | Generate $x_u \sim \text{Uniform}(0, 1)$;
3 | Generate $y_u \sim \text{Uniform}(0, 1)$;
4 | Generate $w(u) \sim \text{Uniform}(w_{min}, w_{max})$;
5 **end**
6 Choose F random demand points from U to place facilities at;
7 Denote facility coordinates with (x_f, y_f), $f \in \{1, \ldots, F\}$;
8 **for** $f \leftarrow 1$ ***to*** F **do**
9 | $S_f \leftarrow \{u \in U : d_{uf} \leq r_{max}\}$;
10 **end**
11 **for** $u \leftarrow 1$ ***to*** $|U|$ **do**
12 | **if** *u is not contained in any set S_f, $f \in \{1, \ldots, F\}$* **then**
13 | | $f^* \leftarrow \underset{f \in \{1,\ldots,F\}}{\arg\min}\ d_{uf}$;
14 | | $S_{f^*} \leftarrow S_{f^*} \cup \{u\}$;
15 | **end**
16 **end**
17 **return** $\mathcal{F} = \{S_1, \ldots, S_F\}$, w

In Algorithm 8, lines 1-5 generate demand points, lines 6-10 generate facilities and their sets and lines 11-16 make sure that every element is in at least one set. These last lines were not included in Resende (1998), which means that some elements were not covered by any set.

Algorithm 8 is only suitable for generating MCP instances, as MCPPM and MCPBC instances require more information. Whenever we generate a MCPPM, we need to determine a partitioning of $\mathcal{F}$ into m disjoint subsets. We consider two partitioning methods: random partitioning and radial partitioning. With random partitioning, every set S_f has an equal probability of $\frac{1}{m}$ to be placed in any partition. With radial partitioning we choose $m = 4$, and we divide the unit square into four equal-sized squares. A set S_f is placed into one of the four partitions based on its coordinates (x_f, y_f). Furthermore, we set the parameters n_i equal to $\lceil \frac{n}{m} \rceil$, where $\lceil \cdot \rceil$ denotes the ceiling function, for all $i \in \{1, \ldots, m\}$.

For the MCPBC we also need to determine a cost function $c : \mathcal{F} \to \mathbb{R}_{\geq 0}$. We consider two methods for determining a cost function: random costs and pay-for-reach costs. In the random cost method, we draw for every set S_f, $f \in \{1, \ldots, F\}$ a random uniform cost with bounds c_{min} and c_{max}. In the pay-for-reach method, we assume that F is an even number. Instead of choosing F random facility locations in line 6, we choose $\frac{F}{2}$ random facility locations and place 2 facilities at each location: one with reach r_{max} and costs c and one with reach $r'_{max} > r_{max}$ and costs $c' > c$. In line 13 of Algorithm 8, instead of considering all the facilities, we place uncovered elements solely into the sets of far-reaching facilities.

For the numerical experiments, we use the values $|U| \in \{100, 150, 200\}$, $F \in \{0.5|U|, 0.8|U|\}$ and $n \in \{0.1F, 0.2F\}$. This gives rise to 12 different parameter combinations. The other parameters are set to $r_{max} = 0.1$, $r'_{max} = 0.2$, $w_{min} = 1$, $w_{max} = 10$, $m = 4$, $c_{min} = 0.5$, $c = 1$ and $c_{max} = c' = 2$. For every parameter combination, we generate 1000 random MCP, MCPPM and MCPBC instances and solve them with the different methods from Section 4. Then, we calculate the performance ratio, which is the ratio of the found objective value to the optimal objective value of the IP. We also calculate the fraction of instances a heuristic returned an optimal solution, as well as the running time of each algorithm.

5.2 Results for the MCP

The numerical results of the heuristics are given in Table 1. This table shows the averages of the statistics for the four parameter cases described in Section 5.1. The four parameter cases themselves are shown in Appendix C. Table 1 reports four statistics for every method: the performance ratio (Perf. Ratio) with standard

deviation, the fraction of optimal solutions (Fr. Opt.) and the average runtime (Runtime). The performance ratio is defined as the average fraction of the objective value found by the heuristic to the optimal objective value (obtained by solving the IP). The fraction of optimal solutions is defined as the fraction of solutions that attain the optimal objective value. Lastly, the runtime is the average runtime of the heuristic, in seconds.

For OblSwap, NonOblSwap and Tabu, we set $k = 1$ in order to limit the runtime. For Tabu the other parameters are $N_I = 0$ and $N_T = 50$. The motivation for these parameters of Tabu is shown in Appendix B. We show in Appendix D results of additional test instances of the MCP.

Table 1: MCP instance results

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9947 (0.0084)	0.6183	0.0014	0.9898 (0.0090)	0.3097	0.0055	0.9854 (0.0094)	0.1345	0.0140
OblSwap	0.9968 (0.0064)	0.6983	0.0036	0.9941 (0.0069)	0.4135	0.0189	0.9916 (0.0072)	0.2173	0.0636
NonOblSwap	0.9798 (0.0199)	0.4123	0.0104	0.9693 (0.0185)	0.1770	0.0686	0.9619 (0.0170)	0.0753	0.2904
Tabu	0.9992 (0.0025)	0.8813	0.1264	0.9975 (0.0040)	0.6450	0.5101	0.9958 (0.0049)	0.4215	1.3430

As we can see from Table 1, all methods are an average performance ratio of over 96%. Of the four tested methods, NonOblSwap performs the worst, as it has the lowest average performance ratio, the highest standard deviation and the lowest fraction of optimal solutions. Tabu performs the best, but its running time is an order of magnitude higher than the other heuristics. We also observe that the performance ratios are decreasing for increasing $|U|$.

As both OblSwap and Tabu take the solution of Greedy and only strictly improve its objective value, it makes sense that OblSwap and Tabu perform better than Greedy. NonOblSwap only strictly increases the auxiliary objective function $w'(\cdot)$. As increasing $w'(\cdot)$ does not necessarily increase $w(\cdot)$, it could be that NonOblSwap actually decreases the weight of the solution provided by Greedy. This is probably the cause of NonOblSwap performing on average the worst of the algorithms.

The tabu list of Tabu forces the local search to continue, even if stuck in a local optimum. This could explain why Tabu performs, on average, better than the other algorithms, as it necessarily enumerates more solutions. The enumeration of more solutions probably also causes the longer runtimes of Tabu, compared to the other algorithms.

5.3 Results for the MCPPM

Tables 2 and 3 report the numerical results for the radial and random partitioning methods for the MCPPM. The separate parameter cases are given in Appendix C, whereas Tables 2 and 3 report the average statistics over the four parameter cases.

For OblSwap, NonOblSwap and Tabu, we set $k = 1$, in order to limit the runtime. For Tabu the other parameters are $N_I = 0$ and $N_T = 50$. The motivation for these parameters of Tabu is shown in Appendix B.

Table 2: Random Partitioning MCPPM instance results

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9801 (0.0188)	0.2068	0.0022	0.9766 (0.0161)	0.0900	0.0081	0.9734 (0.0139)	0.0220	0.0214
OblSwap	0.9833 (0.0166)	0.2405	0.0046	0.9809 (0.0143)	0.1128	0.0197	0.9792 (0.0117)	0.0298	0.0644
NonOblSwap	0.9679 (0.0251)	0.1605	0.0095	0.9596 (0.0214)	0.0730	0.0544	0.9528 (0.0189)	0.0172	0.2132
Tabu	0.9923 (0.0108)	0.4650	0.1403	0.9886 (0.0108)	0.2385	0.5129	0.9860 (0.0096)	0.0915	1.3937

Table 3: Radial Partitioning MCPPM instance results

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9917 (0.0117)	0.5023	0.0023	0.9872 (0.0111)	0.2630	0.0083	0.9821 (0.0110)	0.0763	0.0221
OblSwap	0.9942 (0.0099)	0.5925	0.0048	0.9918 (0.0093)	0.3650	0.0220	0.9890 (0.0089)	0.1563	0.0755
NonOblSwap	0.9798 (0.0201)	0.3462	0.0096	0.9711 (0.0182)	0.1853	0.0601	0.9627 (0.0175)	0.0490	0.2501
Tabu	0.9978 (0.0060)	0.8028	0.1411	0.9957 (0.0066)	0.5590	0.5033	0.9935 (0.0070)	0.3385	1.3728

Tables 2 and 3 show the same pattern as Table 1. Non-oblivious swap local search performs on average the worst and tabu swap local search on average the best. However, Tabu also has a longer runtime than any other algorithm. Furthermore, increasing $|U|$ leads to a decrease of the performance ratio. A difference between the MCP and MCPPM is that the performance ratios and the fraction of optimal solutions for the MCPPM are lower than for the MCP. The runtimes between the MCP and MCPPM do not differ much. Also, the performance ratios of the random partitioning method (Table 2) are lower than of the radial partitioning method (Table 3). However, all methods have a performance ratio higher than 95%.

The same line of reasoning as in Section 5.2 holds. That is, NonOblSwap performs worst because increasing $w'(\cdot)$ does not imply increasing $w(\cdot)$, Tabu performs best as it enumerates more solutions and the performances decrease whenever $|U|$ increases as there are more solutions.

As the radial partitioning method partitions the sets based on their coordinates, sets of the same partition are much more likely to contain the same elements. The facilities (sets) in one partition are spatially nearer to other facilities from the same partition (except for sets on the boundaries). With random partitioning, this is not the case. Thus, radial partitioned MCPPM resembles separate MCP instances more closely than randomly partitioned MCPPM. This could explain the higher performance of the algorithms in Table 3 than in 2.

5.4 Results for the MCPBC

Tables 4 and 5 report the numerical results for the random costs and pay-for-reach instances for the MCPBC. Tables 4 and 5 contain the average statistics over the four problem instances. Again, the separate parameter cases are given in Appendix C.

For OblSwap, NonOblSwap, Tabu, Lagran, TabuRatio and TabuLagran, we set $k = 1$, in order to limit the runtime. As is shown in Appendix B, choosing $N_T = 50$, $L = 50$ and $N_I = 0$ for the random costs and $N_I = 8$ for the pay-for-reach costs seem to yield the highest performances. The parameter κ of CostGreedy is set to 3, as it is the lowest value of κ that achieves an approximation ratio of $(1 - \frac{1}{e})$. For Lagran, the starting value of λ is set to 0 and N_L is set to 50 (see Appendix B). Appendix B also motivates $N_I = 1$ for TabuRatio and TabuLagran. N_T and L are set in both algorithms to 50 to match the values of Tabu.

As the runtime of CostGreedy is several orders of magnitude higher than all the other algorithms, the statistics are only calculated for 100 random instances, instead of 1000. Also, only $|U| = 100$ has been evaluated for this algorithm. The runtimes of TabuLagran include the runtime of Lagran.

Table 4: Random Costs MCPBC instance results

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9833 (0.0151)	0.1923	0.0037	0.9832 (0.0113)	0.0780	0.0143	0.9832 (0.0097)	0.0425	0.0382
OblSwap	0.9925 (0.0106)	0.4560	0.0074	0.9913 (0.0085)	0.2500	0.0299	0.9903 (0.0076)	0.1545	0.0838
NonOblSwap	0.9758 (0.0212)	0.2663	0.0111	0.9659 (0.0190)	0.0832	0.0618	0.9602 (0.0166)	0.0370	0.2403
Tabu	0.9982 (0.0049)	0.7590	0.1425	0.9971 (0.0046)	0.5228	0.5637	0.9965 (0.0044)	0.3620	1.5829
Lagran	0.9966 (0.0063)	0.5905	0.7952	0.9965 (0.0047)	0.3940	3.3518	0.9965 (0.0039)	0.2753	8.7306
TabuRatio	0.9997 (0.0014)	0.9155	0.2082	0.9990 (0.0021)	0.7418	0.9309	0.9983 (0.0026)	0.5535	2.8011
TabuLagran	0.9995 (0.0021)	0.9003	0.9525	0.9992 (0.0021)	0.7528	4.0574	0.9988 (0.0021)	0.5990	10.8570
CostGreedy*	0.9998 (0.0008)	0.9100	194.560	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

Table 5: Pay-for-Reach MCPBC instance results

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9776 (0.0216)	0.3280	0.0030	0.9756 (0.0182)	0.1418	0.0124	0.9743 (0.0140)	0.2523	0.0356
OblSwap	0.9824 (0.0200)	0.3812	0.0060	0.9823 (0.0165)	0.2173	0.0281	0.9814 (0.0129)	0.3170	0.0717
NonOblSwap	0.9662 (0.0280)	0.3022	0.0110	0.9585 (0.0253)	0.1140	0.0749	0.9556 (0.0206)	0.0505	0.3689
Tabu	0.9939 (0.0097)	0.6133	0.1209	0.9933 (0.0092)	0.4538	0.4926	0.9923 (0.0077)	0.4653	1.6451
Lagran	0.9963 (0.0078)	0.7025	0.4930	0.9958 (0.0072)	0.5505	1.2249	0.9946 (0.0064)	0.5203	2.3509
TabuRatio	0.9959 (0.0066)	0.7780	0.1690	0.9938 (0.0095)	0.5750	0.6945	0.9903 (0.0096)	0.5245	1.8167
TabuLagran	0.9987 (0.0039)	0.8647	0.6339	0.9979 (0.0052)	0.7415	1.8451	0.9969 (0.0052)	0.6728	4.0002
CostGreedy*	0.9996 (0.0001)	0.9350	396.526	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

From Table 4 we can see that the performance of TabuRatio and TabuLagran are very similar on average. However, both algorithms perform on average better than all other algorithms, except for CostGreedy. The long runtime makes CostGreedy an undesirable alternative, though. As the TabuLagran also requires Lagran, the runtime of TabuLagran is much higher than TabuRatio. For the random costs MCPBC, this means that TabuRatio is the more desired alternative of the two. We can also see that Tabu performs better than Lagran, despite the larger runtime of Lagran. Just as with the MCP and MCPPM, NonOblSwap has the lowest average performance ratio of all the algorithms. Also, the performance ratio and fraction of optimal solutions decreases whenever $|U|$ increases. All methods have a performance ratio higher than 96%.

The reason for the inferior performance of NonOblSwap is probably again the fact that increasing w' does not guarantee increasing w. The high performance of CostGreedy may be caused by the partial enumeration of the algorithm. This, however, also causes the long runtime. That TabuRatio and TabuLagran perform better than Tabu shows that relaxing the budget constraint and penalizing infeasibilities aids the local search.

From Table 5 we can see that CostGreedy has the highest performance. However, just as in Table 4, the long runtime makes this algorithm undesirable. The algorithm with the highest performance after CostGreedy, is TabuLagran. Unlike in Table 4, TabuRatio performs on average inferior compared to TabuLagran, but the runtime of TabuLagran is still higher than TabuRatio. Another interesting observation is that Lagran has on average a higher performance ratio than Tabu, even though the opposite was true in Table 4. All methods have a performance ratio higher than 95% in Table 5.

Lagran performs relatively better in the pay-for-reach costs than in the random costs, which could explain why TabuLagran performs better than TabuRatio in the pay-for-reach costs MCPBC. Also, as the costs are only $c = 1$ or $c' = 2$ in the pay-for-reach costs, the subgradient could more easily equal 0 for Lagran, explaining the lower runtime of Lagran in the pay-for-reach costs compared to the random costs. A more curious question is why Lagran performs better in the pay-for-reach costs MCPBC than in the ran-

dom costs MCPBC. Investigating several random MCPBC instances reveals that, when using Infeas2Feas, the weight of a solution decreases on average relatively less in a pay-for-reach costs instance than a random costs instance. This in turn means better lower bounds and thus better solutions.

A peculiar result can be observed in the parameter case $|U| = 200$, $F = 0.8|U|$ and $n = 0.2F$ for the pay-for-reach costs MCPBC. The table of this parameter case is duplicated from Appendix C and shown in Table 6. Except for this parameter case, generally the performance ratio and fraction of optimal solutions decreases whenever $|U|$ increases. However, when increasing $|U|$ from 150 to 200 in Table 6, the performance ratios increase drastically and many approximately achieve the optimal ratio of 1.

In this parameter case, the actual parameter values are $|U| = 200$, $F = 160$ and $n = 32$. As there are 80 facilities with radius 0.2 and 80 facilities with radius 0.1, there is a high probability that many solutions cover all elements. The sets (facilities) have a cost either of 1 or of 2. This means that if the optimal solution covers all elements and there are enough sets such that many solutions cover all elements, many feasible optimal solutions exist. This does not happen in the random costs method, as all sets have a radius of 0.1 and the costs are random. That is, even if there are many solutions that cover all elements, many of them can be unfeasible due to the random costs. In the pay-for-reach method with $|U| = 200$, $F = 160$ and $n = 32$, n and F are probably high enough so that many feasible solutions exist that cover all elements (and are therefore optimal). This does explain the sudden increase in performance, as it is easier to construct feasible optimal solutions.

Table 6: Pay-for-Reach MCPBC instance results for $F = 0.8|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9615 (0.0203)	0.0130	0.0060	0.9799 (0.0114)	0.0220	0.0257	0.9994 (0.0016)	0.7950	0.0751
OblSwap	0.9719 (0.0183)	0.0470	0.0138	0.9867 (0.0094)	0.0710	0.0679	0.9997 (0.0010)	0.8960	0.1395
NonOblSwap	0.9323 (0.0278)	0.0030	0.0301	0.9430 (0.0199)	0.0000	0.2252	0.9745 (0.0121)	0.0040	1.1680
Tabu	0.9865 (0.0121)	0.1730	0.2754	0.9946 (0.0059)	0.3060	1.0478	1.0000 (0.0001)	0.9980	3.7611
Lagran	0.9924 (0.0090)	0.3390	1.0301	0.9961 (0.0046)	0.3570	0.9516	1.0000 (0.0000)	0.9990	0.1407
TabuRatio	0.9861 (0.0158)	0.3040	0.4029	0.9937 (0.0081)	0.3980	1.5710	1.0000 (0.0001)	0.9970	3.7617
TabuLagran	0.9965 (0.0070)	0.6060	1.3592	0.9978 (0.0036)	0.5810	2.4267	1.0000 (0.0000)	1.0000	3.5571
CostGreedy*	0.9982 (0.0038)	0.7400	958.298	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

6 Conclusion

In this thesis, we considered three problem variants of the maximum coverage problem: the MCP, the MCPM (more specific the MCPPM) and the MCPBC. For each of these problem variants, we used different heuristics to solve randomly generated problem instances. The algorithms were either greedy algorithms or local search algorithms. We also created a tabu local search algorithm and for the MCPBC, we designed relaxation-based algorithms.

For the MCP and the MCPPM, the Tabu algorithm performs best. Even though NonOblSwap has the highest approximation ratio of the algorithms on those problem instances, it actually has the lowest performance ratio of all algorithms considered. The reason for this low performance is possibly because NonOblSwap does not guarantee a weight increase during its iterations.

Tabu performed the best on the MCP and MCPPM, but its runtime is also the highest. Therefore Tabu is only a viable option if high performance, at the cost of long runtimes, is desired. If not, Greedy and OblSwap are more suitable candidates, as their average performance ratio was always larger than 95%.

For the MCPBC, the CostGreedy algorithm performs best. However, its runtime is several orders of magnitude greater than the other algorithms, which makes the algorithm undesirable. The relaxation-based tabu

swap local search algorithms TabuRatio and TabuLagran perform slightly worse than CostGreedy, but better than all the other algorithms. The performances of TabuLagran and TabuRatio are approximately equal for random costs MCPBC, but the higher runtime of TabuLagran makes TabuRatio preferred in this scenario. For pay-for-reach costs MCPBC, TabuLagran performs better than TabuRatio. This implies that TabuLagran is preferred if sets with higher costs also cover more elements.

However, both TabuRatio and TabuLagran have a runtime several orders of magnitude higher than Greedy, OblSwap and NonOblSwap. Not to mention the dependency of TabuLagran on Lagran makes its runtime longer than TabuRatio. So TabuLagran is only preferred if one wants high performance on MCPBC instances that resemble pay-for-reach costs. If one wants a lower runtime, TabuRatio is the next best option, followed by OblSwap and Greedy. Also TabuLagran is not desired in MCPBC instances with random costs, as TabuRatio performs similar and has a lower runtime.

All considered algorithms, even the ones without an approximation ratio, have a high performance ratio: the average performance ratio was strictly greater than 95%. This implies that simple algorithms as Greedy or OblSwap can be desired if one is willing to sacrifice performance for speed. If not, one could employ the more involved algorithms, such as Tabu, TabuRatio and TabuLagran, to gain some performance at the cost of a longer runtime. This runtime can be several orders of magnitude greater.

Our contribution to this topic is that we have tested several algorithms in concrete maximum coverage problems. We showed that a higher approximation ratio does not guarantee a higher actual performance. This suggests that careful thought should be put into choosing an algorithm for solving maximum coverage problems. This issue is further emphasized by the fact that all the highest-performing algorithms also have the longest runtimes. Furthermore, we introduced two relaxation-based algorithms to solve the budgeted maximum coverage problem. This idea could be further expanded upon by generalizing the relaxed constraint in order to solve maximum coverage problems with more complex constraints. The relaxation-based algorithms could also be modified and analyzed to improve their performance further.

References

Calinescu, G., Chekuri, C., Pál, M., & Vondrák, J. (2011). "Maximizing a monotone submodular function subject to a matroid constraint." *SIAM Journal on Computing*, *40*(6), 1740–1766.

Chekuri, C., & Kumar, A. (2004). "Maximum coverage problem with group budget constraints and applications." In *Approximation, randomization, and combinatorial optimization. algorithms and techniques* (pp. 72–83). Springer.

Feige, U. (1998). "A threshold of ln n for approximating set cover." *Journal of the ACM (JACM)*, *45*(4), 634–652.

Filmus, Y., & Ward, J. (2012). "The power of local search: Maximum coverage over a matroid." In *Stacs'12 (29th symposium on theoretical aspects of computer science)* (Vol. 14, pp. 601–612).

Glover, F. (1989). "Tabu search—part I." *ORSA Journal on computing*, *1*(3), 190–206.

Held, M., & Karp, R. M. (1970). "The traveling-salesman problem and minimum spanning trees." *Operations Research*, *18*(6), 1138–1162.

Held, M., & Karp, R. M. (1971). "The traveling-salesman problem and minimum spanning trees: Part II." *Mathematical programming*, *1*(1), 6–25.

Held, M., Wolfe, P., & Crowder, H. P. (1974). "Validation of subgradient optimization." *Mathematical programming*, *6*(1), 62–88.

Hochbaum, D. S., & Pathria, A. (1998). "Analysis of the greedy approach in problems of maximum k-coverage." *Naval Research Logistics*, *45*(6), 615–627.

Khuller, S., Moss, A., & Naor, J. S. (1999). "The budgeted maximum coverage problem." *Information Processing Letters*, *70*(1), 39–45.

Nemhauser, G. L., Wolsey, L. A., & Fisher, M. L. (1978). "An analysis of approximations for maximizing submodular set functions—I." *Mathematical Programming*, *14*(1), 265–294.

Resende, M. G. (1998). "Computing approximate solutions of the maximum covering problem with GRASP." *Journal of Heuristics*, *4*(2), 161–177.

A Proofs

This section shows various proofs that are referenced in the main text.

Lemma 1. *MCP is a special case of the MCPM*

Proof. The tuple $(\mathcal{F}, \mathbf{F})$ with $\mathbf{F} = \{\mathcal{Y} \subseteq \mathcal{F} : |\mathcal{Y}| \leq n\}$ is the uniform matroid of rank n. Clearly, $\mathbf{F}$ is equal to the collection of feasible sets for the MCP. Therefore, MCP is a special case of the MCPM. □

Lemma 2. *MCP is a special case of the MCPBC*

Proof. For the MCPBC, define $c(S) = 1$ for all $S \in \mathcal{F}$. Then, $\mathbf{F} = \{\mathcal{Y} \subseteq \mathcal{F} : c(\mathcal{Y}) \leq n\} = \{\mathcal{Y} \subseteq \mathcal{F} : \sum_{S \in \mathcal{Y}} c(S) \leq n\} = \{\mathcal{Y} \subseteq \mathcal{F} : \sum_{S \in \mathcal{Y}} 1 \leq n\} = \{\mathcal{Y} \subseteq \mathcal{F} : |\mathcal{Y}| \leq n\}$. As this equals the collection of feasible sets for the MCP, MCP is a special case of MCPBC. □

Lemma 3. *MCPBC is not a special case of MCPM*

Proof. To prove this, we construct a specific instance of the MCPBC and show that $(\mathcal{F}, \mathbf{F})$ is not a matroid. Define $U = \{x, y, z\}$, $S_1 = \{x\}$, $S_2 = \{y\}$ and $S_3 = \{z\}$. Set $w(x) = w(y) = w(z) = 1$, $c(S_1) = c(S_2) = 1$ and $c(S_3) = 3$. Then $\mathbf{F} = \big\{\{\}, \{S_1\}, \{S_2\}, \{S_3\}, \{S_1, S_2\}\big\}$. Let us take $\mathcal{A} = \{S_1, S_2\}$ and $\mathcal{B} = \{S_3\}$. Obviously, $2 = |\mathcal{A}| > |\mathcal{B}| = 1$ and $\mathcal{A}, \mathcal{B} \in \mathbf{F}$. However, there does not exist an $X \in \mathcal{A} \setminus \mathcal{B} = \{S_1, S_2\}$ such that $\mathcal{B} \cup \{X\} \in \mathbf{F}$, as $\{S_1, S_3\} \notin \mathbf{F}$ and $\{S_2, S_3\} \notin \mathbf{F}$. So, $\mathbf{F}$ violates Property 3 of a matroid (Section 3.4) and thus $(\mathcal{F}, \mathbf{F})$ is not a matroid. Therefore, MCPBC is not a special case of MCPM. □

Lemma 4. *MCPM is not a special case of MCPBC*

Proof. To prove this, we construct a specific example of the MCPM. Consider a partition matroid $\mathfrak{m}$ with $\mathcal{F}_1 = \{S_1, S_2\}$, $\mathcal{F}_2 = \{S_1, S_2\}$ and $n_1 = n_2 = 1$. Then,
$\mathbf{F} = \big\{\{\}, \{S_1\}, \{S_2\}, \{S_3\}, \{S_4\}, \{S_1, S_3\}, \{S_2, S_3\}, \{S_1, S_4\}, \{S_2, S_4\}, \big\}$. If we assume a cost function $c : \mathcal{F} \to \mathbb{R}_{\geq 0}$ such that $\mathbf{F} = \{\mathcal{Y} \subseteq \mathcal{F} : c(\mathcal{Y}) \leq n\}$ exists, then it most hold that $c(S_1) + c(S_3) \leq n$ and $c(S_1) + c(S_2) \geq n$. Thus, $c(S_2) \geq c(S_3)$. However, combining this with $c(S_2) + c(S_4) \leq n$, we get $c(S_3) + c(S_4) \leq n$, but $\{S_3, S_4\} \notin \mathbf{F}$. Thus such a function c does not exist and thus MCPM is not a special case of MCPBC. □

Lemma 5. OblSwap *and* NonOblSwap *with* $k = 1$ *achieve approximation ratios of at most 50% on the MCPBC*

Proof. To prove this, we construct a specific instance of the MCPBC. Let us have $U = \{a, b, x\}$ with $w(a) = w(b) = 1$ and $w(x) = \varepsilon > 0$. The sets are defined as $S_1 = \{a, x\}$, $S_2 = \{b, x\}$, $S_3 = \{a\}$ and $S_4 = \{b\}$ with costs $c(S_1) = c(S_2) = 1 + \gamma$ and $c(S_3) = c(S_4) = 1$. The value γ is chosen such that $\frac{1+\varepsilon}{1+\gamma} > 1$ (thus $0 < \gamma < \varepsilon$). The budget is $n = 2$. Clearly, the (cost-adjusted) greedy algorithm gives us the solution $\mathcal{Y} = \{S_1\}$ or $\mathcal{Y} = \{S_2\}$ with weight $1 + \varepsilon$, whereas the optimal solution is $\mathcal{Y}^* = \{S_3, S_4\}$ with weight 2. As every feasible one-exchange from $\mathcal{Y}$ results in a weight decrease (and also a decrease of $f(\mathcal{Y})$), the non-oblivious swap local search cannot improve on the solution found by the greedy algorithm. In this case, the non-oblivious and oblivious swap local search achieves a performance ratio of $\frac{1+\varepsilon}{2}$, which becomes $\frac{1}{2}$ as we can make ε arbitrarily small. This proves thus that, if the non-oblivious and oblivious swap local search has an approximation ratio on the MCPBC, it is at most $\frac{1}{2}$. □

Lemma 6. *If* $\mathcal{X}$ *is the result of an* α*-approximation algorithm to maximize* w^λ*, then* $\frac{1}{\alpha} w^\lambda(\mathcal{X})$ *forms an upper bound on the maximal feasible weight of the MCPBC*

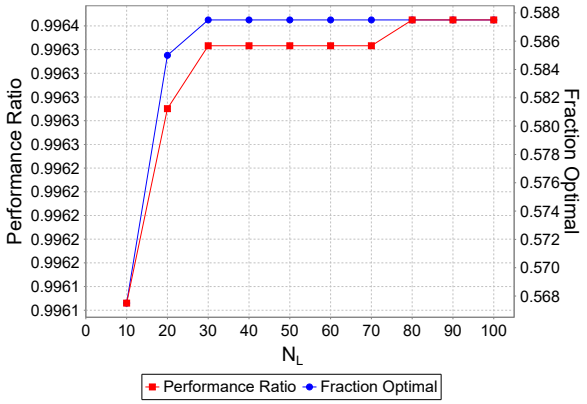
Proof. Let $w^\lambda : 2^{\mathcal{F}} \to \mathbb{R}$ be a non-monotone submodular set function. The function is defined as $w^\lambda(\mathcal{Y}) = w(\mathcal{Y}) + \lambda(n - c(\mathcal{Y}))$. Let us denote $\mathcal{Y}^* = \arg\max_{\mathcal{Y} \in \mathbf{F}} w(\mathcal{Y})$ as the optimal solution to the MCPBC. Furthermore, we define $\mathcal{Y}^\lambda = \arg\max_{\mathcal{Y} \in \mathbf{F}_R} w^\lambda(\mathcal{Y})$ as the optimal solution to the relaxation, given the multiplier λ.

Suppose we have found a solution $\mathcal{X} \in \mathbf{F}_R$ by using an α-approximation algorithm for maximizing w^λ. By definition of an α-approximation algorithm, $l^\lambda(\mathcal{X}) \geq \alpha w^\lambda(\mathcal{Y}^\lambda)$. Furthermore, as $\mathcal{Y}^\lambda$ is the optimal solution, it holds that $w^\lambda(\mathcal{Y}^\lambda) \geq l^\lambda(\mathcal{Y}^*)$. Also, as $\mathcal{Y}^* \in \mathbf{F}$, it means that $c(\mathcal{Y}^*) \leq n$, and thus that $w^\lambda(\mathcal{Y}^*) \geq w(\mathcal{Y}^*)$.

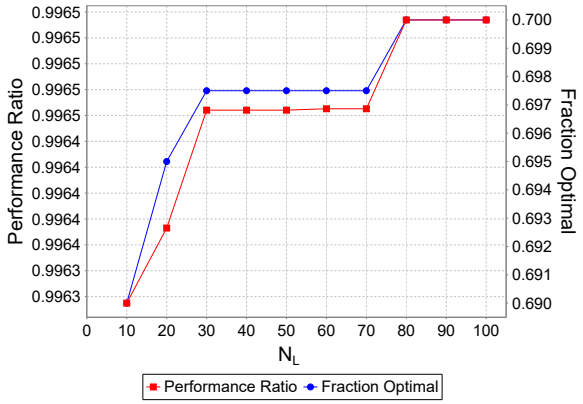
Combining these four inequalities and dividing by α, we get $\frac{1}{\alpha} w^\lambda(\mathcal{X}) \geq w^\lambda(\mathcal{Y}^\lambda) \geq w^\lambda(\mathcal{Y}^*) \geq w(\mathcal{Y}^*)$. Thus $\frac{1}{\alpha} w^\lambda(\mathcal{X})$ forms an upper bound on the maximal feasible weight. □

B Parameter Tuning

This section shows additional graphs and motivations for the chosen parameters of the considered algorithms. We first discuss the parameters for LAGRAN. Then, we discuss the parameters of TABU and its derivatives, TABURATIO and TABULAGRAN. The only parameter of LAGRAN we have to determine is N_L, the number of iterations LAGRAN updates its multiplier before terminating. Figure 1 gives the performance ratios and fractions of optimal solutions of LAGRAN for different values of N_L. These numerical values were obtained by averaging 100 random cost MCPBC instances with $|U| = 100$ and averaging over all four parameter cases (as described in Section 5.1).



(a) $|U| = 100$, random costs MCPBC

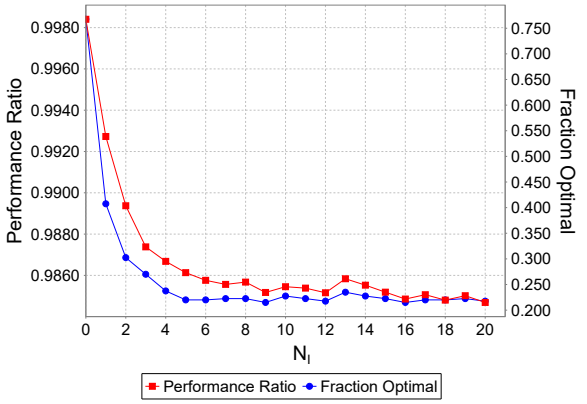


(b) $|U| = 100$, pay-for-reach costs MCPBC

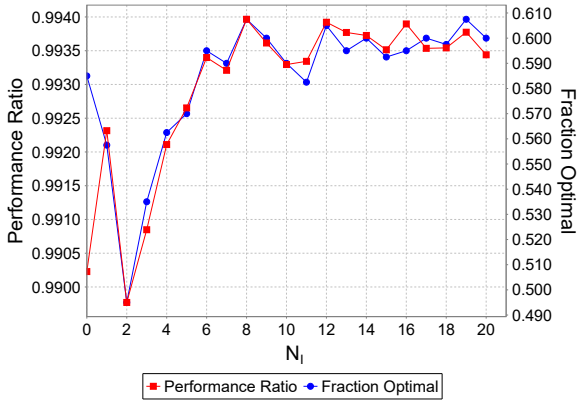
Figure 1: Performance ratios and fractions of optimal solutions of LAGRAN for different values of N_L

As we can observe from Figure 1, whenever N_L is larger than 20, the lines flatten out. Also, as the running time is approximately linear with respect to N_L, we set N_L to 50, compromising performance for speed. In our opinion, the additional performance of increasing N_L to 100 is not worth the extra runtime.

Next, we discuss the parameter settings for TABU. The algorithm has three parameters that need to be tuned: N_I, L and N_T. For the latter, N_T, it seems that a value larger than 10 does not influence the found solutions at all. To be sure N_T is set high enough, we set it to 50. Figure 2 shows the influence of N_I on the MCPBC with $|U| = 100$, averaged over the four parameter cases and 100 random instances each.



(a) $|U| = 100$, random costs MCPBC

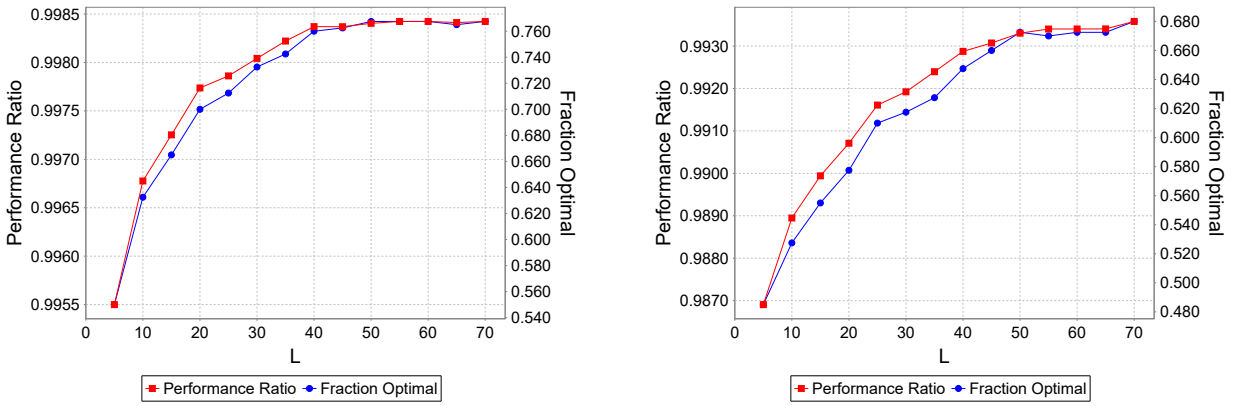


(b) $|U| = 100$, pay-for-reach costs MCPBC

Figure 2: Performance ratios and fractions of optimal solutions of TABU for different values of N_I, with $L = 50$ and $N_T = 50$

Figure 2a shows a very straightforward relation between N_I and performance: the higher I is, the lower the performance is. Therefore we choose $N_I = 0$ for TABU with the random costs MCPBC. This relationship is not at all visible in Figure 2b, however. In fact, it seems the performance increases with N_I. For this reason, we set $N_I = 8$ for TABU with the pay-for-reach costs MCPBC.

Now, we determine the influence of L on the performance of TABU. Figure 3 shows the influence of L on the MCPBC with $|U| = 100$, averaged over the four parameter cases and 100 random instances each. Figure 3 shows the same relationship between L and performance for both the random costs and pay-for-reach costs. That is, the performance increases when L increases and the graphs show diminishing returns. This makes sense, as a larger tabu list forces the tabu search to cycle less and consider more solutions, which could lead to escaping more local optima. As the returns seems to start diminishing if L is larger than 50, we set L equal to 50 for TABU. As this reasoning also holds for TABURATIO and TABULAGRAN, L will be set to 50 for those algorithms as well. Additionally, we set $L = 50$ for TABU on the MCP and MCPM, as the reasoning still holds for these cases.

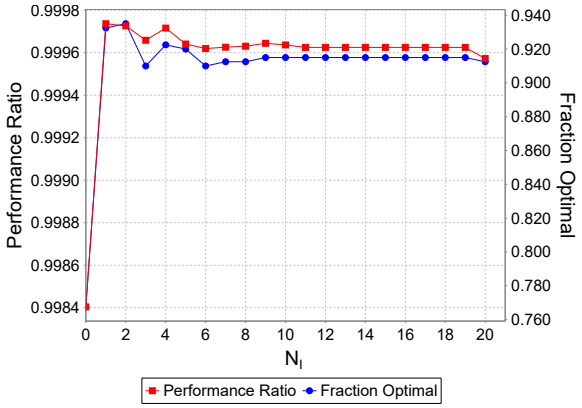


(a) $|U| = 100$, $N_I = 0$, random costs MCPBC

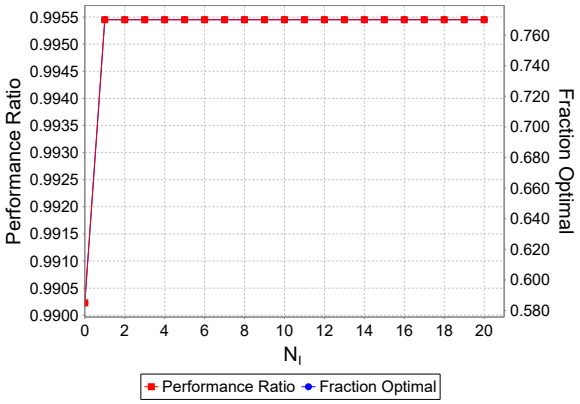
(b) $|U| = 100$, $N_I = 8$, pay-for-reach costs MCPBC

Figure 3: Performance ratios and fractions of optimal solutions of TABU for different values of L, with $N_T = 50$

Now we have determined all the parameter of TABU. Next, we will discuss the parameter settings of TABURATIO and TABULAGRAN. As TABURATIO and TABULAGRAN only differ from TABU in their relaxation, we assume N_T and L can be set to the same values as for TABU. Thus, we set $N_T = 50$ and $L = 50$ for TABURATIO and TABULAGRAN. It is still of interest to investigate the influence of N_I on TABURATIO and TABULAGRAN. Figure 4 shows the performance results of TABURATIO on 100 random problem instances, averaged over the four parameter cases. Likewise, Figure 5 shows the same statistics for TABULAGRAN.

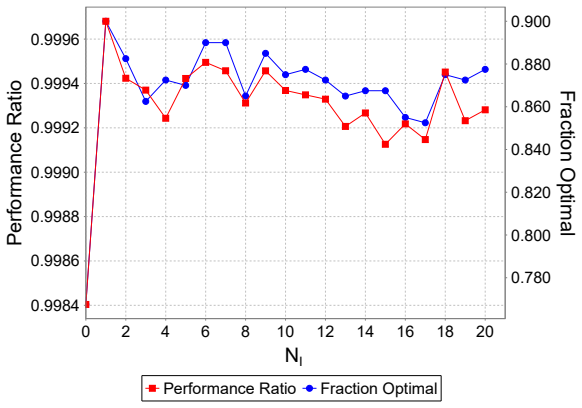


(a) $|U| = 100$, random costs MCPBC

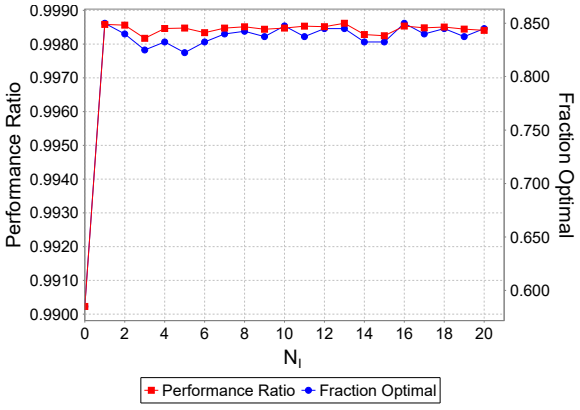


(b) $|U| = 100$, pay-for-reach costs MCPBC

Figure 4: Performance ratios and fractions of optimal solutions of TabuRatio for different values of N_I, with $N_T = 50$ and $L = 50$



(a) $|U| = 100$, random costs MCPBC



(b) $|U| = 100$, pay-for-reach costs MCPBC

Figure 5: Performance ratios and fractions of optimal solutions of TabuLagran for different values of N_I, with $N_T = 50$ and $L = 50$

As can been seen from Figures 4 and 5, choosing $N_I = 0$ leads to inferior performance and the performance stays relatively constant for $N_I > 0$. Choosing $N_I = 0$ renders the relaxation of TabuRatio and TabuLagran ineffective, which reduces these methods back to Tabu. That the relaxation forces the local search to stay close to feasibility probably explains why the performance does not differ much for all $N_I > 0$. As there is no performance increase for $N_I > 1$, we choose $N_I = 1$ for both pay-for-reach costs and random costs and for TabuRatio and TabuLagran. It is worth noting that the choice for N_I is more stable for TabuRatio and TabuLagran than for Tabu, where the cost method influenced the choice of N_I.

C Additional Tables

This section contains the tables of all the numerical results. The column "Perf. Ratio" contains the mean and standard deviation of the performance ratio. "Fr. Opt." denotes the fraction of instances the heuristic returned an optimal solution. "Runtime" denotes the average running time, in seconds.

C.1 Results for the MCP

The following tables summarize all the MCP instance results for the different methods.

Table 7: MCP instance results for $F = 0.5|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9987 (0.0058)	0.9190	0.0004	0.9969 (0.0071)	0.7190	0.0014	0.9942 (0.0082)	0.4420	0.0032
OblSwap	0.9990 (0.0051)	0.9350	0.0009	0.9980 (0.0056)	0.7960	0.0036	0.9961 (0.0066)	0.5680	0.0080
NonOblSwap	0.9957 (0.0132)	0.8240	0.0016	0.9916 (0.0148)	0.5360	0.0067	0.9879 (0.0158)	0.2900	0.0210
Tabu	0.9999 (0.0010)	0.9910	0.0347	0.9996 (0.0021)	0.9450	0.1366	0.9989 (0.0035)	0.8390	0.2867

Table 8: MCP instance results for $F = 0.8|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9956 (0.0093)	0.6820	0.0011	0.9915 (0.0096)	0.2760	0.0043	0.9851 (0.0108)	0.0610	0.0115
OblSwap	0.9971 (0.0072)	0.7430	0.0027	0.9945 (0.0076)	0.4030	0.0111	0.9905 (0.0085)	0.1350	0.0367
NonOblSwap	0.9846 (0.0224)	0.4600	0.0056	0.9768 (0.0203)	0.1180	0.0312	0.9669 (0.0188)	0.0090	0.1317
Tabu	0.9995 (0.0027)	0.9320	0.1010	0.9981 (0.0043)	0.7090	0.3909	0.9958 (0.0057)	0.3800	1.1413

Table 9: MCP instance results for $F = 0.5|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9964 (0.0076)	0.6790	0.0010	0.9911 (0.0092)	0.2370	0.0036	0.9846 (0.0101)	0.0350	0.0094
OblSwap	0.9980 (0.0052)	0.7690	0.0024	0.9952 (0.0065)	0.4050	0.0101	0.9921 (0.0073)	0.1580	0.0339
NonOblSwap	0.9839 (0.0200)	0.3490	0.0047	0.9713 (0.0205)	0.0540	0.0290	0.9581 (0.0183)	0.0020	0.1180
Tabu	0.9997 (0.0016)	0.9350	0.0879	0.9986 (0.0035)	0.7360	0.3321	0.9968 (0.0046)	0.4350	0.8764

Table 10: MCP instance results for $F = 0.8|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9882 (0.0110)	0.1930	0.0031	0.9798 (0.0102)	0.0070	0.0125	0.9776 (0.0085)	0.0000	0.0320
OblSwap	0.9931 (0.0082)	0.3460	0.0085	0.9887 (0.0079)	0.0500	0.0506	0.9876 (0.0064)	0.0080	0.1760
NonOblSwap	0.9547 (0.0240)	0.0160	0.0295	0.9376 (0.0183)	0.0000	0.2074	0.9347 (0.0154)	0.0000	0.8910
Tabu	0.9979 (0.0045)	0.6670	0.2822	0.9938 (0.0062)	0.1900	1.1806	0.9918 (0.0056)	0.0320	3.0679

C.2 Results for the MCPPM - Random Partitioning

The following tables summarize all the random partitioning MCPPM instance results for the different methods.

Table 11: Random Partitioning MCPPM instance results for $F = 0.5|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9847 (0.0192)	0.3450	0.0009	0.9840 (0.0179)	0.2640	0.0017	0.9798 (0.0150)	0.0640	0.0056
OblSwap	0.9864 (0.0179)	0.3770	0.0018	0.9859 (0.0169)	0.3080	0.0035	0.9830 (0.0130)	0.0830	0.0116
NonOblSwap	0.9792 (0.0255)	0.3010	0.0025	0.9798 (0.0222)	0.2450	0.0048	0.9725 (0.0198)	0.0590	0.0216
Tabu	0.9951 (0.0101)	0.6720	0.0589	0.9935 (0.0114)	0.5580	0.1163	0.9905 (0.0100)	0.2460	0.3671

Table 12: Random Partitioning MCPPM instance results for $F = 0.8|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9832 (0.0179)	0.2570	0.0015	0.9795 (0.0147)	0.0520	0.0057	0.9745 (0.0142)	0.0160	0.0158
OblSwap	0.9844 (0.0171)	0.2760	0.0030	0.9817 (0.0139)	0.0720	0.0117	0.9787 (0.0125)	0.0200	0.0360
NonOblSwap	0.9740 (0.0264)	0.2230	0.0044	0.9661 (0.0226)	0.0320	0.0219	0.9574 (0.0207)	0.0100	0.0845
Tabu	0.9925 (0.0119)	0.5130	0.0982	0.9886 (0.0112)	0.1810	0.3648	0.9854 (0.0104)	0.0600	1.0337

Table 13: Random Partitioning MCPPM instance results for $F = 0.5|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9802 (0.0189)	0.1920	0.0017	0.9754 (0.0176)	0.0430	0.0056	0.9702 (0.0158)	0.0080	0.0135
OblSwap	0.9847 (0.0157)	0.2560	0.0034	0.9809 (0.0144)	0.0690	0.0121	0.9774 (0.0129)	0.0160	0.0333
NonOblSwap	0.9702 (0.0237)	0.1130	0.0059	0.9607 (0.0214)	0.0150	0.0261	0.9510 (0.0197)	0.0000	0.0833
Tabu	0.9935 (0.0097)	0.5010	0.1096	0.9897 (0.0103)	0.2020	0.3465	0.9861 (0.0102)	0.0600	0.8421

Table 14: Random Partitioning MCPPM instance results for $F = 0.8|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9724 (0.0193)	0.0330	0.0047	0.9676 (0.0140)	0.0010	0.0193	0.9692 (0.0105)	0.0000	0.0508
OblSwap	0.9778 (0.0159)	0.0530	0.0102	0.9754 (0.0120)	0.0020	0.0513	0.9778 (0.0085)	0.0000	0.1770
NonOblSwap	0.9480 (0.0246)	0.0050	0.0253	0.9320 (0.0193)	0.0000	0.1647	0.9304 (0.0152)	0.0000	0.6637
Tabu	0.9879 (0.0117)	0.1740	0.2945	0.9825 (0.0101)	0.0130	1.2237	0.9820 (0.0079)	0.0000	3.3321

C.3 Results for the MCPPM - Radial Partitioning

The following tables summarize all the radial partitioning MCPPM instance results for the different methods.

Table 15: Radial Partitioning MCPPM instance results for $F = 0.5|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9957 (0.0099)	0.7260	0.0009	0.9949 (0.0102)	0.6500	0.0018	0.9897 (0.0110)	0.2480	0.0058
OblSwap	0.9968 (0.0084)	0.7810	0.0018	0.9963 (0.0088)	0.7210	0.0037	0.9927 (0.0094)	0.3780	0.0123
NonOblSwap	0.9907 (0.0173)	0.5960	0.0023	0.9919 (0.0146)	0.5550	0.0046	0.9832 (0.0173)	0.1760	0.0221
Tabu	0.9991 (0.0047)	0.9370	0.0592	0.9988 (0.0053)	0.8920	0.1214	0.9966 (0.0066)	0.6480	0.3727

Table 16: Radial Partitioning MCPPM instance results for $F = 0.8|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9928 (0.0132)	0.6140	0.0015	0.9887 (0.0122)	0.2320	0.0060	0.9820 (0.0123)	0.0400	0.0166
OblSwap	0.9943 (0.0119)	0.6690	0.0030	0.9917 (0.0105)	0.3430	0.0125	0.9880 (0.0103)	0.1100	0.0408
NonOblSwap	0.9877 (0.0194)	0.5090	0.0040	0.9789 (0.0196)	0.1390	0.0239	0.9686 (0.0190)	0.0180	0.1004
Tabu	0.9982 (0.0069)	0.8720	0.0982	0.9963 (0.0071)	0.5920	0.3724	0.9935 (0.0082)	0.3150	1.0444

Table 17: Radial Partitioning MCPPM instance results for $F = 0.5|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9933 (0.0106)	0.5030	0.0018	0.9881 (0.0108)	0.1650	0.0058	0.9818 (0.0116)	0.0170	0.0143
OblSwap	0.9957 (0.0083)	0.6260	0.0035	0.9932 (0.0087)	0.3530	0.0132	0.9903 (0.0085)	0.1290	0.0396
NonOblSwap	0.9808 (0.0203)	0.2470	0.0056	0.9715 (0.0198)	0.0470	0.0272	0.9620 (0.0185)	0.0020	0.0952
Tabu	0.9986 (0.0047)	0.8600	0.1099	0.9969 (0.0059)	0.6210	0.3498	0.9950 (0.0064)	0.3720	0.8528

Table 18: Radial Partitioning MCPPM instance results for $F = 0.8|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9851 (0.0131)	0.1660	0.0049	0.9770 (0.0113)	0.0050	0.0195	0.9747 (0.0094)	0.0000	0.0517
OblSwap	0.9901 (0.0110)	0.2940	0.0107	0.9863 (0.0093)	0.0430	0.0587	0.9852 (0.0075)	0.0080	0.2092
NonOblSwap	0.9600 (0.0233)	0.0330	0.0265	0.9422 (0.0188)	0.0000	0.1844	0.9369 (0.0153)	0.0000	0.7828
Tabu	0.9954 (0.0075)	0.5420	0.2968	0.9909 (0.0082)	0.1310	1.1723	0.9890 (0.0066)	0.0190	3.2215

C.4 Results for the MCPBC - Random Costs

The following tables summarize all the random costs MCPBC instance results for the different methods.

Table 19: Random Costs MCPBC instance results for $F = 0.5|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9814 (0.0206)	0.3330	0.0012	0.9839 (0.0141)	0.1780	0.0047	0.9845 (0.0125)	0.1280	0.0111
OblSwap	0.9945 (0.0121)	0.7060	0.0022	0.9945 (0.0089)	0.5020	0.0090	0.9940 (0.0086)	0.4120	0.0211
NonOblSwap	0.9889 (0.0199)	0.5840	0.0025	0.9829 (0.0185)	0.2520	0.0106	0.9787 (0.0175)	0.1370	0.0270
Tabu	0.9988 (0.0056)	0.9080	0.0384	0.9982 (0.0045)	0.7660	0.1489	0.9977 (0.0048)	0.6730	0.3242
Lagran	0.9962 (0.0089)	0.7460	0.1881	0.9968 (0.0058)	0.5920	0.9410	0.9968 (0.0051)	0.5210	2.4557
TabuRatio	1.0000 (0.0005)	0.9920	0.0528	0.9998 (0.0012)	0.9640	0.2318	0.9997 (0.0016)	0.9140	0.5599
TabuLagran	0.9997 (0.0022)	0.9690	0.2281	0.9995 (0.0021)	0.9030	1.1239	0.9994 (0.0020)	0.8530	2.9034
CostGreedy*	1.0000 (0.0000)	1.000	30.0219	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

Table 20: Random Costs MCPBC instance results for $F = 0.8|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9836 (0.0148)	0.1830	0.0040	0.9830 (0.0115)	0.0610	0.0157	0.9822 (0.0097)	0.0180	0.0432
OblSwap	0.9931 (0.0104)	0.4700	0.0074	0.9917 (0.0089)	0.2360	0.0294	0.9903 (0.0080)	0.0940	0.0830
NonOblSwap	0.9775 (0.0232)	0.2340	0.0090	0.9677 (0.0204)	0.0400	0.0422	0.9604 (0.0174)	0.0040	0.1398
Tabu	0.9980 (0.0051)	0.7670	0.1239	0.9966 (0.0055)	0.5100	0.4417	0.9958 (0.0050)	0.3030	1.2534
Lagran	0.9964 (0.0064)	0.5860	0.7307	0.9964 (0.0047)	0.3910	3.2392	0.9961 (0.0041)	0.2300	9.5979
TabuRatio	0.9998 (0.0016)	0.9580	0.1843	0.9992 (0.0021)	0.7920	0.7496	0.9983 (0.0029)	0.5550	2.2899
TabuLagran	0.9995 (0.0023)	0.9160	0.8742	0.9993 (0.0022)	0.7920	3.8369	0.9988 (0.0025)	0.6320	11.4837
CostGreedy*	0.9998 (0.0012)	0.9300	572.952	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

Table 21: Random Costs MCPBC instance results for $F = 0.5|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9858 (0.0132)	0.2050	0.0021	0.9848 (0.0109)	0.0700	0.0078	0.9840 (0.0091)	0.0240	0.0212
OblSwap	0.9936 (0.0099)	0.4940	0.0044	0.9922 (0.0087)	0.2470	0.0166	0.9902 (0.0077)	0.1100	0.0461
NonOblSwap	0.9805 (0.0196)	0.2330	0.0054	0.9702 (0.0195)	0.0410	0.0248	0.9618 (0.0170)	0.0070	0.0879
Tabu	0.9987 (0.0042)	0.8250	0.0903	0.9978 (0.0041)	0.5930	0.3221	0.9967 (0.0043)	0.3850	0.8976
Lagran	0.9972 (0.0055)	0.6300	0.4891	0.9970 (0.0043)	0.4210	2.0385	0.9969 (0.0035)	0.2800	5.9021
TabuRatio	0.9999 (0.0010)	0.9630	0.1261	0.9995 (0.0018)	0.8640	0.5045	0.9987 (0.0026)	0.6230	1.4959
TabuLagran	0.9997 (0.0016)	0.9310	0.5818	0.9995 (0.0017)	0.8360	2.4268	0.9992 (0.0020)	0.6990	7.0486
CostGreedy*	0.9999 (0.0005)	0.9500	67.2553	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

Table 22: Random Costs MCPBC instance results for $F = 0.8|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9822 (0.0117)	0.0480	0.0073	0.9813 (0.0088)	0.0030	0.0291	0.9823 (0.0074)	0.0000	0.0773
OblSwap	0.9887 (0.0099)	0.1540	0.0154	0.9869 (0.0077)	0.0150	0.0647	0.9868 (0.0063)	0.0020	0.1849
NonOblSwap	0.9561 (0.0221)	0.0140	0.0277	0.9430 (0.0174)	0.0000	0.1698	0.9400 (0.0144)	0.0000	0.7064
Tabu	0.9973 (0.0047)	0.5360	0.3174	0.9959 (0.0043)	0.2220	1.3423	0.9957 (0.0035)	0.0870	3.8566
Lagran	0.9966 (0.0043)	0.4000	1.7727	0.9960 (0.0038)	0.1720	7.1887	0.9962 (0.0029)	0.0700	16.9669
TabuRatio	0.9990 (0.0024)	0.7490	0.4692	0.9976 (0.0031)	0.3470	2.2378	0.9965 (0.0032)	0.1220	6.8586
TabuLagran	0.9991 (0.0025)	0.7850	2.1256	0.9985 (0.0025)	0.4800	8.8420	0.9978 (0.0021)	0.2120	21.9926
CostGreedy*	0.9994 (0.0016)	0.7600	1078.12	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

C.5 Results for the MCPBC - Pay-for-Reach

The following tables summarize all the pay-for-reach costs MCPBC instance results for the different methods.

Table 23: Pay-for-Reach MCPBC instance results for $F = 0.5|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9915 (0.0199)	0.7110	0.0010	0.9843 (0.0203)	0.4180	0.0038	0.9792 (0.0199)	0.2080	0.0101
OblSwap	0.9921 (0.0192)	0.7320	0.0016	0.9879 (0.0187)	0.5270	0.0063	0.9846 (0.0179)	0.3330	0.0168
NonOblSwap	0.9901 (0.0225)	0.6990	0.0019	0.9817 (0.0249)	0.3820	0.0081	0.9753 (0.0245)	0.1950	0.0252
Tabu	0.9980 (0.0083)	0.9020	0.0328	0.9972 (0.0088)	0.8230	0.1346	0.9954 (0.0088)	0.6260	0.3461
Lagran	0.9979 (0.0081)	0.8950	0.1265	0.9978 (0.0075)	0.8420	0.5391	0.9959 (0.0084)	0.6530	1.3839
TabuRatio	1.0000 (0.0002)	0.9990	0.0372	0.9995 (0.0038)	0.9680	0.1577	0.9977 (0.0079)	0.8470	0.4379
TabuLagran	1.0000 (0.0002)	0.9990	0.1622	0.9992 (0.0048)	0.9420	0.6734	0.9984 (0.0058)	0.8420	1.7422
CostGreedy*	1.0000 (0.0000)	1.0000	32.1906	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

Table 24: Pay-for-Reach MCPBC instance results for $F = 0.8|U|$ and $n = 0.1F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9797 (0.0233)	0.3210	0.0032	0.9704 (0.0216)	0.0860	0.0128	0.9565 (0.0187)	0.0050	0.0373
OblSwap	0.9828 (0.0219)	0.3860	0.0052	0.9772 (0.0200)	0.1540	0.0220	0.9670 (0.0181)	0.0220	0.0738
NonOblSwap	0.9742 (0.0305)	0.2940	0.0068	0.9597 (0.0291)	0.0570	0.0362	0.9370 (0.0249)	0.0030	0.1485
Tabu	0.9959 (0.0090)	0.7100	0.0973	0.9914 (0.0115)	0.3930	0.4296	0.9856 (0.0123)	0.1360	1.3785
Lagran	0.9971 (0.0077)	0.7790	0.4390	0.9944 (0.0091)	0.5150	1.8861	0.9905 (0.0098)	0.2300	4.8168
TabuRatio	0.9985 (0.0062)	0.9060	0.1285	0.9907 (0.0135)	0.4620	0.5662	0.9784 (0.0166)	0.0970	1.6447
TabuLagran	0.9989 (0.0053)	0.9170	0.5483	0.9967 (0.0074)	0.6820	2.3580	0.9930 (0.0095)	0.3950	6.3422
CostGreedy*	1.0000 (0.0000)	1.0000	535.766	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

Table 25: Pay-for-Reach MCPBC instance results for $F = 0.5|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9779 (0.0230)	0.2670	0.0018	0.9678 (0.0196)	0.0410	0.0071	0.9619 (0.0157)	0.0010	0.0201
OblSwap	0.9829 (0.0206)	0.3600	0.0035	0.9773 (0.0180)	0.1170	0.0162	0.9745 (0.0148)	0.0170	0.0569
NonOblSwap	0.9684 (0.0313)	0.2130	0.0053	0.9494 (0.0271)	0.0170	0.0298	0.9356 (0.0209)	0.0000	0.1340
Tabu	0.9953 (0.0093)	0.6680	0.0782	0.9901 (0.0107)	0.2930	0.3580	0.9882 (0.0097)	0.1010	1.0948
Lagran	0.9978 (0.0064)	0.7970	0.3764	0.9950 (0.0077)	0.4880	1.5228	0.9922 (0.0074)	0.1990	3.0621
TabuRatio	0.9990 (0.0043)	0.9030	0.1073	0.9913 (0.0126)	0.4720	0.4908	0.9852 (0.0139)	0.1570	1.4226
TabuLagran	0.9994 (0.0033)	0.9370	0.4659	0.9980 (0.0050)	0.7610	1.9223	0.9964 (0.0056)	0.4540	4.3598
CostGreedy*	1.0000 (0.0000)	1.0000	60.8512	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

Table 26: Pay-for-Reach MCPBC instance results for $F = 0.8|U|$ and $n = 0.2F$

	$\|U\| = 100$			$\|U\| = 150$			$\|U\| = 200$		
	Perf. Ratio	Fr. Opt.	Runtime (s)	Perf. Ratio	Fr. Op.	Runtime (s)	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9615 (0.0203)	0.0130	0.0060	0.9799 (0.0114)	0.0220	0.0257	0.9994 (0.0016)	0.7950	0.0751
OblSwap	0.9719 (0.0183)	0.0470	0.0138	0.9867 (0.0094)	0.0710	0.0679	0.9997 (0.0010)	0.8960	0.1395
NonOblSwap	0.9323 (0.0278)	0.0030	0.0301	0.9430 (0.0199)	0.0000	0.2252	0.9745 (0.0121)	0.0040	1.1680
Tabu	0.9865 (0.0121)	0.1730	0.2754	0.9946 (0.0059)	0.3060	1.0478	1.0000 (0.0001)	0.9980	3.7611
Lagran	0.9924 (0.0090)	0.3390	1.0301	0.9961 (0.0046)	0.3570	0.9516	1.0000 (0.0000)	0.9990	0.1407
TabuRatio	0.9861 (0.0158)	0.3040	0.4029	0.9937 (0.0081)	0.3980	1.5710	1.0000 (0.0001)	0.9970	3.7617
TabuLagran	0.9965 (0.0070)	0.6060	1.3592	0.9978 (0.0036)	0.5810	2.4267	1.0000 (0.0000)	1.0000	3.5571
CostGreedy*	0.9982 (0.0038)	0.7400	958.298	- (-)	-	-	- (-)	-	-

*: Calculated with 100 observations.

D Results for the Test MCP Instances

This section gives the results of 50 test instances of the MCP. These instances are generated with a similar algorithm as Algorithm 8. For these instances, 250 random points are generated on the unit square as demand locations. The weight of each point is uniformly generated with bounds 0 and 1. Furthermore, we place a facility at each demand point, resulting in 250 facilities. Each facility covers another demand point if the Euclidean distance is smaller than some predefined radius. This radius is chosen such that 90% of the demand points are covered in the optimal (IP) solution of the problem. Finally, the maximum cardinality n of a solution is 15.

These instances can be generated with Algorithm 8, by choosing the parameters $|U| = 250$, $F = 250$, $n = 15$, $w_{min} = 0$ and $w_{max} = 1$. As mentioned before, r_{max} is chosen such that the optimal solution covers 90% of the demand points. The numerical results of these 50 instances are given in Table 27

Table 27: Test MCP instance results

	Perf. Ratio	Fr. Opt.	Runtime (s)
Greedy	0.9649 (0.0123)	0.0000	0.0277
OblSwap	0.9821 (0.0108)	0.0000	0.1919
NonOblSwap	0.9462 (0.0237)	0.0000	0.4903
Tabu	0.9918 (0.0071)	0.1000	3.8735