The Maximum Facility Location Problem

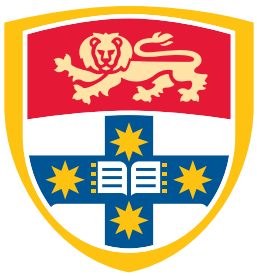
MITCHELL JONES

Supervisor: Dr. Julián Mestre

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Computer Science and Technology (Advanced) (Honours)

School of Information Technologies
The University of Sydney
Australia

3 November 2015



THE UNIVERSITY OF
SYDNEY

Abstract

In the maximum facility location problem, we are given a set of clients C and facilities F. Assigning a client u to a facility v incurs a cost w_{uv} and each facility has an associated interval I_v on the real line. Our goal is to assign as many clients to open facilities in order to maximise the cost of assignment, subject to the constraint that open facilities cannot have their respective intervals overlap.

In this thesis we design two new approximation algorithms for this problem that use LP-rounding techniques. The first is 0.19-approximation algorithm, and the second is a randomised rounding algorithm. Afterwards, we turn our attention of the hardness of the problem, and show that the maximum facility location problem cannot be approximated in polynomial time within a factor of $(1 - 1/e + \varepsilon)$ for some $\varepsilon > 0$ unless P = NP.

In order to demonstrate the utility of our algorithm, we apply it to a biological scenario in which we aim to map sequences of nucleotides (reads) from a given donor genome to a reference genome. We show that our algorithms are up to two order of magnitudes faster than existing algorithms, without a sacrifice in the quality of results.

Acknowledgements

First and foremost, I would like to thank Dr. Julián Mestre for his expert knowledge on the subject, continued support and encouragement throughout this project.

I'd like to thank both Dr. Stefan Canzar and Dr. Khaled Elbassioni for their help and contributions to this project. In particular Dr. Stefan Canzar, for his great explanations in helping me understand the biological nature of the problem, providing me with data sets (produced by CLEVER) and help interpreting the results.

CONTENTS

List of Figures

List of Tables

Notation

F	The set of facilities.
C	The set of clients.
u	Some client in C.
v	Some facility in F.
w_{uv}	The cost of assigning a client u to a facility v.
$w(u)$	The facility v that a client u can be assigned to with the most weight w_{uv}.
f	A submodular function.
I_v	The interval on the real line associated with a facility v.
P	The set of all interval endpoints.
x_{uv}	A client-facility variable that is 1 if a client u is assigned to a facility v, otherwise 0. Takes a real value in the range $[0, 1]$ if the variable is in a linear program.
y_v	A facility variable that is 1 if a facility v is open, otherwise 0. Takes a real value in the range $[0, 1]$ if the variable is in a linear program.
(x, y)	A solution to the integer/linear program.
slack(p)	The slack across a point $p \in P$ in the integer/linear program. The point p is tight if slack$(p) = 0$.
$A + a$	Shorthand notation for $A \cup \{a\}$, where A is a set and a is an element.
$A - a$	Shorthand notation for $A \backslash \{a\}$, where A is a set and a is an element.

CHAPTER 1

Introduction

The facility location problem is a classical NP-hard problem in Computer Science. Given a set of facilities F and clients C, our goal is to assign every client to a facility. For each client $u \in C$ and facility $v \in F$, there is a connection cost w_{uv}. In addition, there is a cost o_v for opening a facility $v \in F$. Therefore our goal is to minimise the quantity $\sum_{v \in S} o_v + \sum_{u \in C} \min_{v \in S} w_{uv}$, where $S \subseteq F$ are the subset of facilities we chose to open.

The facility location problem has been studied for many years, with one of the first approximation algorithms developed by Shmoys et al. (1997). In their work, the authors develop a 4-approximation algorithm. They apply a common technique which will be seen throughout the rest of this thesis. First, they model the problem as an *integer program*. Integer programs allow one to mathematically model a problem, where the goal is to optimise some objective function subject to a set of constraints. Unfortunately, it's not surprising that integer programs are NP-hard to solve. However, one can instead relax the model to a *linear program*. This means the variables in the model take on real values in the range $[0, 1]$ – they admit a fractional solution. However it doesn't necessarily make sense to open 0.5 of a facility, and so we must *round* this fractional solution back to an integral solution, while still ensuring the solution obeys the constraints of the model. More recent results have appeared since the work by Shmoys et al. (1997). An example of this is the paper by Li (2013), where he develops a 1.488-approximation algorithm for the facility location problem. This algorithm currently yields the best known approximation ratio.

In this thesis we look at a modified version of the facility location problem, which we call the *maximum facility location problem*. Firstly, we impose that $o_v = 0$ for all $v \in F$ and aim to maximise our objective instead of minimising it. Finally, we say that each facility $v \in F$ has an interval I_v on the real line, and open facilities cannot have their respective intervals overlap.

This means we aim to select an independent set of facilities $S \subseteq F$ to maximise the quantity $\sum_{u \in C} \max_{v \in S} w_{uv}$.

We present a biological application of this problem, which is related to genome mapping. For some species, researchers have been able to construct the full reference genome. However, this reference genome is never exactly representative of a real (donor) genome. There may be many differences such as mutations, sequences of nucleotides (reads) that have been inserted or deleted and so on. Therefore it is of interest to map these reads from the donor genome back to the provided reference genome. We show how our abstract problem can be applied in this scenario, and compare the performance of our algorithms against existing methods.

1.1 Contribution

In this work we develop two new approximation algorithms for the maximum facility location problem. The central idea in both algorithms is to solve a relaxed version of the integer program formulation of this problem. The algorithms differ in how they round this fractional solution back to an integral solution.

Formally, we say that a polynomial time approximation algorithm is a ρ-approximation, if for all instances of the problem produces a solution whose value is at least within ρ of the optimal solution. In particular for maximisation problems, we select a $\rho < 1$. Another way to phrase it is "This algorithm never produces a value less than ρ times the optimal." For minimisation problems, $\rho > 1$. In this case we can interpret the definition as "This algorithm never produces a value that is greater than ρ times the optimal."

We show the first algorithm results in a 0.19-approximation. The high level idea is to randomly select a set of facilities, and then remove conflicts. This ensures that the final set of facilities are independent. The second algorithm is a rounding algorithm, where we randomly modify the fractional values until we obtain integral values. This method results in a non-constant approximation algorithm. Afterwards, we develop inapproximability results for the maximum facility location problem. More specifically, we show that the problem can not be approximated better than a factor of $(1 - 1/e)$ in polynomial time unless P = NP.

Finally, we apply our algorithms to the problem of genome mapping. We compare our methods to previously known methods on simulated and real data sets. Our algorithms are up to two orders of magnitude faster than existing methods, without sacrificing solution quality (precision and recall).

1.2 Outline

To begin, we provide a background of the relevant material that will be needed for the rest of this thesis. Such as a brief introduction to approximation and randomised algorithms and the rounding techniques that will be used in our algorithms (Chapter 2). Afterwards, we formally define the maximum facility location problem, and develop the integer (and linear) program (Chapter 3).

Following the formulation, we discuss the two new algorithms and analyse their various properties and approximation ratio. We also develop a hardness result for our problem (Chapter 4). In the next chapter, we look at a biological application of this abstract problem. Our methods are experimented with on simulated and real data sets. At the end of the chapter we discuss the results (Chapter 5).

We finish this thesis by providing a final discussion of the overall results (Chapter 6) and make some concluding remarks (Chapter 7).

CHAPTER 2

Background

Optimisation problems are an important field of Computer Science and Mathematics. Any problem that can be modelled as an objective function with (optional) constraints is an optimisation problem.

To begin, we will start with the definition and motivation behind approximation and randomised algorithms. Afterwards, we will look at a class of functions known as *submodular* functions. Various types of maximisation problems will be discussed. In the section following, we will discuss the various types of techniques which are used to approximate a solution to these problems.

2.1 Approximation & Randomised Algorithms

We start by discussing the concept of approximation and randomised algorithms. This will provide the appropriate background material before looking at the other concepts in the review.

Many problems in Computer Science are typically NP-hard. These problems are either *decision* or *optimisation* problems. Decision problems involve returning a "yes" or "no" solution to a problem. A common example is the subset-sum problem: Given a set of integers $x_1, \ldots, x_n$, does there a exist a subset of these integers that add up to a target C? On the other hand, optimisation problems ask to maximise or minimise some objective. An example is the maximum independent set. Given a graph $G = (V, E)$ where V are the set of vertices and E are the set of edges, we say two vertices are independent if they do not share an edge. The problem asks to find a subset of vertices $S \subseteq V$ that are independent, such that we maximise $|S|$. This is where the idea of an *approximation* algorithm comes into play. While we may not have access

to polynomial time solutions for some problems, we can derive solutions that run in polynomial time which provide a guarantee on the solution.

Consider the set cover problem. We are given a universal set of items U and m subsets $S_1, S_2, \ldots S_m$ such that $\forall i \in [1, m] : S_i \subseteq U$. In addition, each set S_i has a non-negative weight w_i. We want to select a set C of these subsets S_i such that $\cup_{S \in C} S = U$. Our goal is to minimise the total weight, $\sum_{S \in C} w_S$. Johnson (1973), Lovász (1975) and Chvatal (1979) all independently present a solution which produces a $H_{d^\star}$-approximation, where H_n is the nth harmonic number[1] and $d^\star$ is the size of the largest subset. The algorithm is simple, at each step, we simply pick the set S that minimises the quantity $\frac{w_S}{|S \backslash \cup_{T \in C} T|}$. Intuitively, we want to select sets with a low weight that cover a lot of new elements.

A branch of approximation algorithms are *randomised* algorithms. Randomised algorithms involve making random choices in order to reach a final solution. One can then analyse the expected value of the solution and derive an approximation ratio. A problem that can be approximated with a surprisingly simple randomised algorithm is the MAX-SAT problem. In the MAX-SAT problem, we are given n boolean variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$ in disjunctive normal form. Each clause also has an associated weight w_j for $j \in [1, m]$. Our goal is to set the values of x_i so that we maximise the weighted sum of the satisfied clauses. Johnson (1973) was the first to introduce an algorithm for the MAX-SAT problem. Afterwards, Yannakakis (1994) developed a randomised variant of Johnson's algorithm, which resulted in a $\frac{1}{2}$-approximation algorithm. The approach is to simply set each variable x_i to true with probability $\frac{1}{2}$. This leads to a $\frac{1}{2}$-approximation for the MAX-SAT problem.

2.2 Set Function Optimisation Problems

In this section we discuss the concept of submodular functions and their properties. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n elements and 2^S be the *power set* of S. A set function $f : 2^S \to \mathbb{R}$ maps each subset of S to a real value. A function is *submodular* if $\forall A \subseteq B \subseteq S$ and $\forall x \in S \backslash B$

$$f(A + x) - f(A) \geq f(B + x) - f(B).$$

[1] Formally, the nth harmonic number H_n is defined as $H_n = \sum_{k=1}^{n} \frac{1}{k}$.

In other words, the difference in adding an element x to a smaller set is larger than adding x to a bigger set. Additionally, a set function f is *non-decreasing* if for any $A \subseteq B \subseteq S$

$$f(B) \geq f(A).$$

Nemhauser et al. (1978) list various sorts of functions and prove their submodularity. More precisely, given an integer K, they propose the following problem for any submodular function f

$$\max_{A \subseteq U} \{\, f(A) \mid |A| \leq K, f \text{ is submodular} \,\}.$$

This is known as the submodular maximisation problem subject to a cardinality constraint, where we aim to select a subset of (at most) K elements from U such that we maximise f.

2.2.1 Matroid Constraints

Let $M = (U, I)$, where U is a universal set of items and I is a collection of subsets of U. Then M is a *subset system* if for any $T \subset S \subseteq U$, we have that if $S \in I$, then this implies $T \in I$.

The system M is a *matroid* if it has the *matroid property*. The matroid property states that for any two subsets $S, T \in I$, where $|S| < |T|$, then there must exist an element $x \in T \backslash S$ such that $S + x \in I$.

Nemhauser et al. (1978) also consider maximising submodular functions subject to a matroid constraint. Given a set $U' \subseteq U$, let $D(U') = \{\, S \in I \mid S \subseteq U' \,\}$. In other words, $D(U')$ is the set of all subsets that are contained within U', that are independent. Finally, let each element $i \in U$ have a cost c_i. It is then shown that the function

$$f(U') = \max_{F \in D(U')} \sum_{i \in F} c_i$$

is submodular and non-decreasing.

Fisher et al. (1978) focus on maximising a submodular function f over P intersecting matroids (U, I_p) for $p \in [1, P]$. Formally, they define the problem as

$$\max_{A \subseteq U} \left\{ f(A) \mid A \in \bigcap_{p=1}^{P} I_p \right\}.$$

2.2.2 Knapsack Constraints

Another common problem is maximising a (not necessarily monotone) submodular function f over a knapsack constraint. The knapsack constraint is defined as follows. Each element $i \in U$ has a cost c_i and we're given a number $\mathcal{C}$. Our goal is to pick a subset $A \subseteq U$ of elements maximising $f(A)$ such that $\sum_{i \in A} c_i \leq \mathcal{C}$ (Kulik et al., 2013).

An extension of this problem is to maximise f subject to multiple knapsack constraints. Formally, consider the problem of maximising f subject to d knapsack constraints (for a fixed d). This means for each knapsack constraint $j \in [1, d]$, each element has a cost $c_{i,j}$ and an upperbound $\mathcal{C}_j$ (Kulik et al., 2013). This means we want to solve

$$\max_{A \subseteq U} \left\{ f(A) \mid \sum_{i \in A} c_{i,j} \leq \mathcal{C}_j \quad \forall j \in [1, d] \right\}.$$

2.2.3 Independent Set in Interval Graphs

The last problem we will consider is the following problem: Given a set of intervals U on the real line, and a submodular function f, we wish to select a subset of intervals $A \subseteq U$ such that we maximise $f(A)$. In addition, we impose the constraint that each pair of intervals in A cannot overlap. Formally, the aim is to solve

$$\max_{A \subseteq U} \{ f(A) \mid \text{No two intervals in } A \text{ overlap} \}$$

This problem is studied by Feldman (2013) and will be discussed later in more detail.

2.3 Algorithm Techniques

Now that all of the main problems have been defined, we spend the rest of the chapter looking at the main algorithms and techniques used to solve these problems. First, we discuss some basic greedy algorithms used to maximise a submodular function subject to the various constraints defined (such as cardinality and matroid constraints).

Next, we will look at an algorithm known as the continuous greedy algorithm. This algorithm is used to improve the approximation ratios for maximising submodular functions subject to a matroid constraint and subject to a knapsack constraint.

Afterwards, the concept of a contention resolution scheme is introduced. Using this, we show how one can use these schemes to obtain an approximation algorithm for the the problem of maximising a submodular function by selecting an independent set of intervals in an interval graph.

Finally, we will discuss a technique known as pipage rounding. The goal of this technique is to round the fractional solution provided by a linear program to an integral solution. To illustrate pipage rounding, we discuss a simple rounding algorithm for solving maximum weight matching over a bipartite graph.

2.3.1 Greedy Algorithms

In this section we will discuss some greedy algorithms used to solve the problems mentioned in Section 2.2. Consider the problem of finding a set $A \subseteq U$ that maximises $f(A)$ such that $|A| \leq K$ for some integer K. Nemhauser et al. (1978) present the following approximation algorithm. For K iterations, select the element x that yields the largest increase. In other words, we aim to find an element x such that $x = \arg\max_{x' \in U \setminus A} f(A + x') - f(A)$. We then add x to A and continue. This rather simple algorithm results in a $\left(1 - \frac{1}{e}\right)$-approximation.

An alternate interesting algorithm proposed by the authors is the R-interchange heuristic. The algorithm works as follows. We pick an arbitrary set S such that $|S| = K$. We then try to find a new set S' such that $|S - S'| \leq R$ (i.e. the difference between the two sets is at most R) and $f(S') > f(S)$. We then continue this to find another set S'', and so on. This iterative algorithm terminates when we cannot find a new set that improves the value. The algorithm is a $\left(\frac{K}{2K-R}\right)$-approximation.

Another problem discussed was maximising a submodular function f subject to the matroid constraint, as defined in Section 2.2.1. Fisher et al. (1978) proposed a greedy algorithm to solve this problem. Given the initial universe U, at each step we add an element $x \in U$ to a set S that maximises the difference $f(S + x) - f(S)$. If this element x is in the intersection of the P matroids (i.e. $x \in \cap_{p=1}^{P} I_p$), then add x to S and continue. Otherwise, we set $U = U - x$ and repeat the previous step (now finding an element $x' \in U - x$ that maximises the difference). This process is repeated until U is empty. this heuristic also leads to a $\left(\frac{1}{P+1}\right)$-approximation.

This means if we want maximise a submodular function over a single matroid constraint, then the algorithm described is a $\frac{1}{2}$-approximation.

2.3.2 Continuous Greedy

We now focus our attention on a different type of greedy algorithm, known as the *continuous greedy* algorithm. Given a submodular (possibly non-monotone) function f, we would like to maximise f. In the cases when there are additional constraints (such as the matroid constraint), they typically lead to worse approximations as they are harder to solve. Gharan and Vondrák (2011) describe an algorithm that provided a 0.41-approximation algorithm for unconstrained submodular maximisation and a 0.325-approximation when subjected to the matroid constraint.

In order to discuss their algorithm, we first introduce the *multilinear extension* of a submodular function f. Recall that given a universal set of elements U, f is defined as $f : 2^U \to \mathbb{R}$. Given a vector $x = [0, 1]^{|U|}$, let $R \subseteq U$ be a random set in which element i is in R with probability x_i. A multilinear extension of f is a function $F : [0, 1]^U \to \mathbb{R}$ that is the expected value of f given R. Formally,

$$F(x) \triangleq \mathbb{E}[f(R)] = \sum_{S \subseteq U} f(S) \prod_{i \in S} x_i \prod_{i \notin S} (1 - x_i). \tag{2.1}$$

Simulated annealing is a popular heuristic used to search for the maximum (or minimum) of a function. The idea behind simulated annealing is to start bouncing around the feasible region and gradually settle down into a maximum (or minimum) point. The reason we start by randomly moving around the feasible region is to avoid being trapped in a local maximum (or minimum). Gharan and Vondrák (2011) present an algorithm which is based on simulated annealing. Given a set $S \subset U$ and a parameter $p \in [0, 1]$, they define a probability distribution $\mathcal{R}_p(S)$ by starting from S and adding/removing each element independently with probability $1 - p$. Let 1_S be the characteristic vector of a set $S \subseteq U$ (coordinate $i = 1$ if and only if $i \in S$). They observe that $\mathbb{E}[f(\mathcal{R}_p(S))] = F(p1_S + (1 - p)1_{\bar{S}})$ (where $\bar{S} = U \backslash S$, the compliment of S). Therefore when $p = 1/2$, this is equivalent to bringing elements into S by flipping a coin. As $p \to 1$, little modifications to S are made at all.

Therefore the algorithm starts as follows. Given an initial empty set S and $p = \frac{1}{2}$, we continuously add elements to S if there exists an element $i \in U$ such that $F(p1_{S+i} + (1 - p)1_{\bar{S}-i}) >$

$F(p1_S + (1-p)1_{\bar{S}})$. We then increment p by some small value δ and continue while $p < 1$. Afterwards, we return the best solution among *all* sets (either S or $\bar{S}$) encountered. In order for the analysis to hold, $\delta = n^{-3}$. While this algorithm is easy to implement, it does not run in polynomial time. This is because we need to constantly evaluate the multilinear extension F (2.1) in order to find an element $i \in U$ that improves the solution at each step. In addition, the algorithm performs at least $O(n^3)$ iterations, which is not very practical.

As discussed in Section 2.3.1, there exists a $\left(\frac{1}{P+1}\right)$-approximation algorithm for maximising f over the intersection of P matroids. For example, when we just have a single matroid, we have a $\frac{1}{2}$-approximation. With the results by Gharan and Vondrák (2011), Calinescu et al. (2011) were able to improve the approximation factor to $1 - \frac{1}{e} > \frac{1}{2}$, but they estimate the running time of the implementation would be $O(n^8)$. This running time is dominated by the number of times the algorithm would have to 'query' the membership oracle (to test if a set $A \subseteq S$ is in I). More precisely, the authors say it would require $O(n^7)$ queries to the membership oracle, which is hardly practical. The goal is to approximate $\max\{F(y) \mid y \in P(M)\}$, where $P(M)$ is the polytope[2] of a matroid M. After finding a solution $y \in [0,1]^U$, we round it to an integral solution y' such that $y' \in \{0,1\}^U$ and $f(y') \geq F(y) \geq \left(1 - \frac{1}{e}\right) \times \text{OPT}$ (where OPT is the value of the optimal solution).

Calinescu et al. (2011) present their modified version of the continuous greedy algorithm. Let $\delta = \frac{1}{9d^2}$, where d is the *rank* of the matroid.[3] Again, we start with an empty solution y and this time $t = 0$. Let R be a set where an element i is added with probability y_i. Since we cannot evaluate F directly (as it would be expensive iterating through $O(2^{|U|})$ subsets), we estimate F by sampling. At each iteration, we perform $\frac{10}{\delta^2}(1 + \ln n)$ samples of R to estimate $w_i = \mathbb{E}[f(R+j) - f(R)]$ for each $i \in U$. These values w_i are the weight of each element i. We then find a set J maximising the sum of the weights in the matroid M and set $y = y + \delta 1_J$. We then increment t by δ, and terminate when $t \geq 1$, returning y. The main reason we sample F so many times is to ensure that we get a good estimate with high probability. In fact, with high probability this algorithm yields a $\left(1 - \frac{1}{e} - \frac{1}{3d}\right)$-approximation. However the authors state that with a small enough δ, the last term is eliminated.

[2] A polytope is an n-dimensional object with flat sides. For example, a 2-dimensional polytope is a polygon, and a 3-dimensional polytope is a polyhedron.

[3] The rank of a matroid $M = (U, I)$ is the size of the largest set in I. Formally, $r(U) = \max_{S \subseteq U : S \in I} |S|$.

As defined by Section 2.2.2, Kulik et al. (2013) consider the problem of maximising f subject to d knapsack constraints (for a fixed d). This means for each knapsack constraint $j \in [1, d]$, each element has a cost $c_{i,j}$ and an upperbound $\mathcal{C}_j$. We will concentrate on the case when f is monotone, in which they claim a (nearly optimal) $\left(1 - \frac{1}{e} - \varepsilon\right)$-approximation for any $\varepsilon > 0$. The authors show that the problem can be solved with an $(\alpha - \varepsilon)$-approximation, if there is a polynomial time α-approximation algorithm for the continuous problem with respect to f. This is done by using the continuous greedy algorithm and then performing rounding. Since f is monotone, we know of a $\left(1 - \frac{1}{e}\right)$-approximation. Therefore this leads us to a $(1 - \frac{1}{e} - \varepsilon)$-approximation as claimed.

2.3.2.1 Faster Versions of Continuous Greedy

The reader may have noticed that we previously mentioned that the running time of the continuous greedy algorithm is very large. This is mostly due to the small δ steps made at each iteration. Commonly there is a trade-off between computation time and an approximation ratio. In this section, we discuss some algorithms developed by Badanidiyuru and Vondrák (2014) and Wei et al. (2014) that study modified versions of the continuous greedy algorithm which are faster in practice, but lead to a slightly worse approximation ratio.

In Section 2.3.1 we discussed a simple greedy algorithm that selects elements that improve the solution if they form one of the subsets in I. This yielded a $\frac{1}{2}$-approximation. On the other hand, we also discussed the continuous greedy algorithm which results in a $\left(1 - \frac{1}{e}\right)$-approximation. The insight made by Badanidiyuru and Vondrák (2014) is to introduce an intermediate value $\delta \in (0, 1]$ that interpolates between these two solutions. The solution they present results in a $\left(1 - \frac{1}{(1+\delta)^{1/\delta}}\right)$-approximation. Therefore if $\delta = 1$, then we get the original greedy algorithm. As $\delta \to 0$, we approach the other $\left(1 - \frac{1}{e}\right)$-approximation.

In the first algorithm discussed, we select individual elements and add them to our set S. Compared to the continuous greedy algorithm where we estimate F and select an entirely new set of elements J at each iteration. Therefore they provide an algorithm which is a combination between these two solutions, choosing a set amount of items at each iteration and then updating the solution instead of picking an entire independent set (or only a single element). This reduces the number of samples needed to estimate the multilinear extension F, but also the number of

time steps (depending on the value of δ). The algorithm they derive ultimately results in a $\left(1 - \frac{1}{e} - \varepsilon\right)$-approximation algorithm for some $\varepsilon > 0$.

In Section 2.2, we also discussed the problem of maximising a submodular function f such that the set $S \subseteq U$ we pick has at most K elements. Once again, greedily picking elements based on the largest benefit led to a $\left(1 - \frac{1}{e}\right)$-approximation. However, when f is complex to evaluate the algorithm can still be time consuming. Wei et al. (2014) propose various algorithms to speed up the greedy solution. The first they discuss in the *approximate greedy* algorithm. In an iteration of the normal greedy algorithm, given our current set S we select an element $x^\star$ such that $x^\star = \arg\max_{x \in U} f(S + x) - f(S)$. Instead, the lazy greedy algorithm only selects an element x' that is within β times the benefit of adding $x^\star$ (for some $\beta < 1$). This means that the number of times needed to evaluate f can be reduced. The trade-off is in the approximation. Having this β parameter means that the lazy greedy algorithm is only a $\left(1 - \frac{1}{e^\beta}\right)$-approximation.

The second algorithm they propose is called *multi greedy*. The purpose of this algorithm is to reduce the complexity of evaluating f by breaking it down into multiple *proxy* submodular functions (Wei et al., 2014). With each next function requiring more computation to evaluate than the previous (the authors even propose the idea that the last proxy function could be the function f itself). Furthermore, if this is combined with the described approximate greedy algorithm, then there will be an overall reduction in computation. For each of these γ proxy functions, we also define a set of cardinality constraints $\{\ell_1, \ldots, \ell_\gamma\}$, such that $\sum_{i=1}^{\gamma} \ell_i = K$. For each of the γ proxy functions, they run the greedy algorithm and return the union over the final sets. The authors observed that $\gamma = 2$ was even enough to notice a performance increase.

2.3.3 Contention Resolution Schemes

In this section we discuss schemes for maximising f subject to common constraints. For this section we use the following notation. If (U, I) is a subset system, then let $P(I)$ be a polytope of the convex relaxations imposed by I. Given a solution $x \in P(I)$, let R denote a set of elements added with probability x_i. A *c-balanced contention resolution* scheme for $P(I)$ involves reducing R to a set S such that $S \in I$, and an element x_i that is in R is in S with probability c. Formally, $\Pr[x_i \in S \mid x_i \in R] \geq c$. This holds for any $c > 0$. However, if we need to work with a larger value of c, then we can 'scale down' $P(I)$ by some factor $b \in [0, 1]$. In other words, we define $bP(I) = \{\, b \cdot x \mid x \in P(I) \,\}$. This is known as a (b, c)-balanced contention resolution (CR)

scheme. Let π be a function that given this random set R produces a set $S \in I$. Given a solution $x \in bP(I)$, π is a (b, c)-balanced CR scheme if $\Pr[x_i \in \pi(R) \mid x_i \in R] \geq c$ for all x_i. Finally, we say the balanced CR scheme is monotone if $\Pr[x_i \in \pi(A)] \geq \Pr[x_i \in \pi(B)]$ for some $A \subseteq B$.

Using resolution schemes that are (b, c)-balanced is attractive because it leads to the following theorem. For a given solution $x \in bP(I)$ and a set $S = \pi(R)$, we have that $\mathbb{E}[f(S)] \geq cF(x)$.

Vondrák et al. (2011) also prove the existence of certain CR schemes for various constraints. For example, given a matroid $M = (U, I)$ (where $n = |U|$), they prove that there exists a $\left(b, \frac{1-(1-b/n)^n}{b}\right)$-balanced CR scheme.

The other constraint studied by Vondrák et al. (2011) is the knapsack constraint. As before, let each item $i \in U$ have a cost c_i. We want to select a set $S \subseteq U$ such that we maximise $f(S)$ and $\sum_{i \in S} c_i \leq \mathcal{C}$. In this instance, we set $\mathcal{C} = 1$ by scaling down the values of c_i. Then Vondrák et al. (2011) show that for any $b \in (0, \frac{1}{2})$, there is a monotone $(b, 1 - 2b)$-balanced CR scheme. In addition, they prove a stronger result. If $c_i \leq \delta$ for all $i \in U$ and $\delta \in (0, \frac{1}{2})$, then, for any $b \in (0, \frac{1}{2e})$, there exists a monotone $(b, 1 - (2eb)^{(1-\delta)/\delta})$-balanced CR scheme. In addition, if there is an $\varepsilon \in (2\delta, \frac{1}{2})$ then there is a $(1 - \varepsilon, 1 - e^{-\Omega(\varepsilon^2/\delta)})$-balanced CR scheme.

While the theorems are attractive, they are existential. This means we do not know the exact scheme itself, but rather only it's values of b and c. Some examples of schemes are proposed by Feldman (2013). An example we will discuss is maximising a submodular function f over an interval graph. Given a set of intervals U, a solution $S \subseteq U$ is feasible if and only if no two intervals overlap. Therefore we would like to maximise $f(S)$ subject to this constraint. Feldman presents a $\frac{1}{4}$-approximation to this problem.

To do this, Feldman uses a modified version of the continuous greedy algorithm as discussed earlier. His observation was that the continuous greedy algorithm chooses the next set that maximises the increase in adding an element x to the set. Instead, Feldman proposes that we should choose the next element according to the gradient of F. This means that we take smaller steps, and allow ourselves to have a stopping time t of larger than 1. Feldman shows that this algorithm still produces a solution inside the polytope $P(I)$ and has a $(1 - e^{-t} - o(1))$-approximation. This is known as the *measured* continuous greedy algorithm.

Therefore, Feldman uses the measured version with a stopping time of $t = \ln 2$ to produce a solution $x \in \ln 2 \cdot P(I)$. Following the definition of the (b, c)-balanced CR scheme, this implies $b = t = \ln 2$ also. We then create a random set R, where an element x_i is in R with probability x_i. The scheme Feldman uses is the following. For every interval $u \in U$, we mark u for deletion if there is another interval $u' \in R$ that intersects the start point of u. We then remove all marked intervals from R and return this new set S. One can show that $\Pr[u \in S \mid u \in R] = e^{-b}$ for all $u \in R$. Since we know that $F(x) \geq (1 - e^{-t} - o(1)) \times \text{OPT}$, this means that the algorithm is a $\mathbb{E}[f(S)] \geq e^{-b}F(x) \geq e^{-b}(1 - e^{-b} - o(1)) \times \text{OPT}$ approximation. Finally, by setting $b = \ln 2$, we see that this is a $\frac{1}{4}$-approximation as claimed.

2.3.4 Pipage Rounding

The previous section discussed taking a fractional solution $x \in [0, 1]^U$ and performing operations on it to obtain a valid (integral) solution. In this section we will discuss another rounding technique. Compared with CR schemes, which select elements with probability x_i and then perform some sort of conflict resolution, this method of rounding involves working directly on the solution x in order to reach an integral solution $\hat{x} \in \{0, 1\}^U$.

The first technique we will discuss is *pipage rounding*. Introduced by Ageev and Sviridenko (2004), it involves rounding a fractional solution x to an integral one given a set of constraints. In most optimisation solutions, there are constraints on the solution x. As an example to demonstrate the technique, consider the following problem. Given a bipartite graph $G = (A, B; E)$ where (A, B) is the bipartition and a weight w_e on each edge e, we want to select a matching to maximise the sum of these weights. More formally, we wish to select a matching M to maximise $\sum_{e \in M} w_e$. For each edge $e \in E$, we will have a variable $x_e \in \{0, 1\}$, where $x_e = 1$ if we choose this edge and 0 otherwise. However, since we aim to find a fractional solution, we allow $x_e \in [0, 1]$. For a node $u \in A \cup B$, let $\delta(u) = \{ (u, v) \in E \}$ be the edges adjacent to u. We can construct the following linear program.

$$\begin{aligned} \text{maximise} \quad & \sum_{e \in E} w_e x_e \\ \text{subject to} \quad & \sum_{e \in \delta(u)} x_e \leq 1 \quad \forall u \in A \cup B \\ & x_e \geq 0 \quad \forall e \in E \end{aligned} \tag{LP}$$

Our goal is then to *round* the fractional solution $x \in [0,1]^{|E|}$ to an integral solution $x \in \{0,1\}^{|E|}$. First, we split the set of edges with a non-integral value into two matchings M_1 and M_2. Pipage rounding works by decreasing (or increasing) every edge in M_1 by some small ε (or δ) amount, while increasing (or decreasing) every edge in M_2 by the same ε (or δ) amount in order to preserve any constraints. Gandhi et al. (2006) propose the following method. Let $S = \{\, e \in E \mid 0 < x_e < 1 \,\}$ be the set of edges with a fractional value in (LP). One can prove that S will either have a cycle, or be a simple path. In either case, we can split a path or cycle into two matchings M_1 and M_2. We can then define two new variables

$$\varepsilon = \min\left(\min_{e \in M_1} x_e, \min_{e \in M_2}(1 - x_e)\right),$$

and

$$\delta = \min\left(\min_{e \in M_1}(1 - x_e), \min_{e \in M_2} x_e\right).$$

Essentially, ε represents how much we can increase the fractional value of every edge in M_2, and decrease the fractional value of every edge in M_1. Conversely, δ represents the case where we decrease the fractional value of every edge in M_2 and increase the fractional value of every edge in M_1.

With probability $\frac{\delta}{\varepsilon+\delta}$, define a new solution x'

$$x'_e = \begin{cases} x_e - \varepsilon, e \in M_1 \\ x_e + \varepsilon, e \in M_2 \\ x_e, \text{ otherwise} \end{cases}.$$

Otherwise with probability $\frac{\varepsilon}{\varepsilon+\delta}$, define a new solution x'

$$x'_e = \begin{cases} x_e + \delta, e \in M_1 \\ x_e - \delta, e \in M_2 \\ x_e, \text{ otherwise} \end{cases}$$

This means that at each iteration, we will make at least one edge in our fractional solution x take on an integral value. An attractive thing about rounding is that the expected value of x'

does not change. For example, consider all edges $e \in M_1$, then we have

$$\begin{aligned}\mathbb{E}[x'_e] &= \left(\frac{\varepsilon}{\varepsilon+\delta}\right)(x_e - \delta) + \left(\frac{\delta}{\varepsilon+\delta}\right)(x_e + \varepsilon) \\ &= \frac{\varepsilon x_e - \varepsilon\delta + \delta x_e + \varepsilon\delta}{\varepsilon+\delta} = x_e.\end{aligned}$$

A more general rounding algorithm is presented by Chekuri et al. (2010). They describe an algorithm known as the randomised swap rounding scheme. Essentially, they start by expressing a fractional solution x as a *convex combination*[4] of the vertices of $P(I)$. Therefore, $x = \sum_{i=1}^{n} \lambda_i v_i$ where each v_i is a vertex of $P(I)$ and λ is a vector of scalars chosen.

The next step is called the merge operation. Start at $w_1 = v_1$ and $\beta_1 = \lambda_i$. At each iteration we 'merge' the vertices w_i and v_{i+1} and set $\beta_{i+1} = \beta_i + \lambda_{i+1}$ such that $\mathbb{E}[\beta_{i+1} w_{i+1}] = \beta_i w_i + \lambda_{i+1} v_{i+1}$. Then over all iterations we produce a rounded solution $x' = w_n$ such that $\mathbb{E}[x'] = \sum_{i=1}^{n} \lambda_i v_i = x$. Essentially, this algorithm performs a random walk from vertex to vertex. However the exact details of this algorithm lie in this merge procedure.

As an example, they provide a merge algorithm for using the randomised swap rounding method on matroids. As before, we find a fractional solution x and decompose it into a convex combination such that $x = \sum_{i=1}^{n} \lambda_i v_i$. At an iteration i, we wish to merge w_i and v_{i+1}. They propose the following method. While $w_i \neq v_{i+1}$, we pick an $a \in w_i \backslash v_{i+1}$ and $b \in v_{i+1} \backslash w_i$ such that $w_i - a + b \in I$ and $v_{i+1} - b + a \in I$.[5] Now, with probability $\frac{\beta_i}{\beta_i + \lambda_{i+1}}$ we let $v_{i+1} = v_{i+1} - b + a$. Otherwise we let $w_i = w_i - a + b$. We then finally return w_i.

Notice that this form of randomly determining which set to modify (either w_i or v_{i+1}) in expectation doesn't change the value of the solution. It is this similar technique used by Gandhi et al. (2006) that allows one to assert that $\mathbb{E}[x'] = x$ with this method.

[4] A convex combination of a set of n vectors $\{v_i\}$ is the sum $\sum_{i=1}^{n} \lambda_i v_i$ for some scalars $\lambda_i \geq 0$ such that $\sum_{i=1}^{n} \lambda_i = 1$.

[5] Since w_i and v_{i+1} are vertices of the polytope $P(I)$, we denote $w_i - a$ as setting the a coordinate of w_i to 0. Similarly for $w_i + a$.

CHAPTER 3

Preliminaries

In this chapter we formally define our abstract optimisation problem, which we call the *maximum facility location problem*, and later give a linear relaxation that will be the basis of our algorithms.

3.1 Maximum Facility Location

Let C be a set of clients and F be a set of facilities. Let $w : C \times F \to \mathbb{R}$ be the weight of assigning a given client to a given facility. Associated with every facility $v \in F$ is an interval I_v on the real line.

DEFINITION 1. A subset of facilities $S \subseteq F$ is *independent* if no two intervals in the set $\{ I_v : v \in S \}$ overlap.

Finally, we use P to denote the set of endpoints of all intervals.

DEFINITION 2. For any non-empty subset $S \subseteq F$, the *value* of S is given by

$$f(S) = \sum_{u \in C} \max_{v \in S} w_{uv}. \tag{3.1}$$

For completeness, we define $f(\varnothing) = 0$.

An instance of the maximum facility location problem is specified by a tuple (C, F, w, I). The objective is to choose an independent subset $S \subseteq F$ maximising $f(S)$. See Figure 3.1.

We now prove an important property about this objective function.

DEFINITION 3. A function f is submodular if $\forall A \subseteq B \subseteq F$ and $\forall v \in F \setminus B$

$$f(A + v) - f(A) \geq f(B + v) - f(B).$$

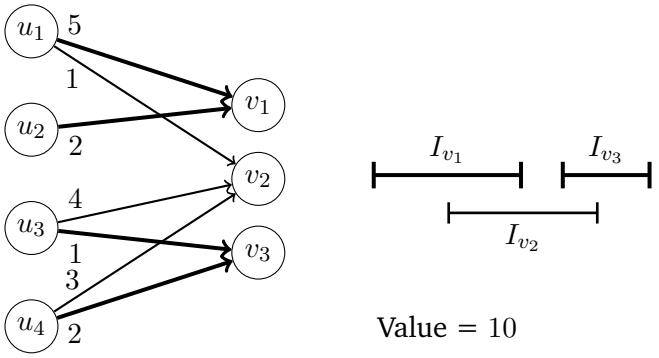


FIGURE 3.1: Example instance of the maximum facility location problem, with $C = \{u_1, u_2, u_3, u_4\}$ and $F = \{v_1, v_2, v_3\}$. The set of facilities we choose to open must be independent. Therefore we can either pick $\{v_2\}$ or $\{v_1, v_3\}$. If we open v_2, then the value of our solution is 8. If we open v_1 and v_3, then we get a better solution (bold edges) with value 10.

LEMMA 1. *The function f defined in* (3.1) *is monotone non-decreasing and submodular.*

PROOF. Let $A \subseteq B \subseteq F$. Notice that $\max_{v' \in A} w_{uv'} \leq \max_{v' \in B} w_{uv'}$ for all $u \in C$. Summing over all $u \in C$ we get $f(A) \leq f(B)$. Therefore, f is monotone non-decreasing.

To prove the submodularity of f, consider a facility $v \in F \setminus B$. We add v to both A and B and analyse the gain in value for some arbitrary client $u \in C$:

$$\begin{aligned}\max_{v' \in A+v} w_{uv'} - \max_{v' \in A} w_{uv'} &= \max(0, w_{uv} - \max_{v' \in A} w_{uv'}), \\ &\geq \max(0, w_{uv} - \max_{v' \in B} w_{uv'}) = \max_{v' \in B+v} w_{uv'} - \max_{v' \in B} w_{uv'}.\end{aligned}$$

Summing over all $u \in C$ we get $f(A+v) - f(A) \geq f(B+v) - f(B)$. Therefore, f is submodular.

□

3.2 Integer Program

Our integer program (IP) uses two type of variables. For each facility $v \in F$, we have a *facility variable* y_v that indicates whether facility v is open. For each client $u \in C$ and facility $v \in F$, we have a *client-facility variable* x_{uv} that captures whether u is assigned to v.

The integer program has three types of constraints. For each client $u \in C$ we have a constraint enforcing that u is assigned to at most one facility. For each client $u \in C$ and facility $v \in F$ we

have a constraint enforcing that if u is assigned to v then v is open. Finally, for each interval endpoint $p \in P$ we have a constraint enforcing that we choose at most one of the facilities whose intervals span p.

$$\begin{aligned}
\text{maximise} \quad & \sum_{u \in C, v \in F} w_{uv} x_{uv} \\
\text{subject to} \quad & \sum_{v \in F} x_{uv} \leq 1 && \forall u \in C \\
& x_{uv} \leq y_v && \forall u \in C, v \in F \\
& \sum_{v \in F : p \in I_v} y_v \leq 1 && \forall p \in P \\
& x_{uv} \in \{0,1\} && \forall u \in C, v \in F \\
& y_v \in \{0,1\} && \forall v \in F
\end{aligned} \tag{IP}$$

3.3 Linear Relaxation

Solving an integer program is NP-hard, and thus for large instances can take an extremely long time to solve. While there are general heuristics such as the branch and bound method (Land and Doig, 1960), in the worst case they still take exponential time.

To combat this problem, we turn our integer program (IP) in a linear program (LP) by *relaxing* the variables. Instead of each variable taking a value in the set $\{0, 1\}$, we allow it to take any real value in the range $[0, 1]$. While it doesn't make sense to "open half a facility," or "assign a client to a third of a facility," linear programs are much faster to solve. Our goal for the rest of this thesis will then be designing algorithms that turn this fractional solution into an integral solution.

$$\begin{aligned}
\text{maximise} \quad & \sum_{u \in C, v \in F} w_{uv} x_{uv} \\
\text{subject to} \quad & \sum_{v \in F} x_{uv} \leq 1 && \forall u \in C \\
& x_{uv} \leq y_v && \forall u \in C, v \in F \\
& \sum_{v \in F : p \in I_v} y_v \leq 1 && \forall p \in P \\
& x_{uv} \geq 0 && \forall u \in C, v \in F \\
& y_v \geq 0 && \forall v \in F
\end{aligned} \tag{LP}$$

CHAPTER 4

Algorithms

Now that we have defined the problem formally and stated the linear program, in this chapter we will devise two new algorithms. Both of these algorithms involve solving (LP) to obtain a fractional solution (x, y) and rounding to obtain an integral solution. The first is a 0.19-approximation algorithm based off the work on Feldman and his contention resolution scheme for maximising a submodular function subject over an interval graph (Feldman, 2013). The second is a dependent rounding algorithm, based on the technique of pipage rounding as discussed in Chapter 2, Section 2.3.4.

4.1 Independent Rounding

The first algorithm we present very simple. First, we compute an optimal fractional solution (x, y) of the linear relaxation (LP). Then we create a set $R \subseteq F$ by placing each facility $v \in F$ into R with probability αy_v, where $\alpha \in [0, 1]$ is a parameter that will be chosen later to maximise the approximation ratio in our analysis. Finally, we apply a filtering step that drops some facilities so as to obtain an independent set $T \subseteq R$.

Algorithm 1 Independent rounding with alterations.

function SELECT-AND-FILTER(α)
  $(x, y) \leftarrow$ solve (LP)
  $R \leftarrow$ Select each $v \in F$ with probability αy_v
  $T \leftarrow \{ v \in R \mid \nexists\, v' \in R$ such that start point of I_v lies in $I_{v'} \}$
  return T
end function

It should be noted that the alteration step (where we go from R to T) is the same as the conflict resolution scheme of Feldman (2013) that was developed for the more general problem

of maximising a submodular function subject to the constraint that the set chosen is independent in an auxiliary interval graph. As mentioned in Chapter 2, while this algorithm is a 0.25-approximation, it is not practical as it requires that we solve the stronger multilinear relaxation (see Equation (2.1) of Section 2.3.2) rather than the much easier linear relaxation (LP).

There is an intuition behind randomly selecting facilities with probability αy_v. If some facility $v \in F$ is assigned a value y_v that is close to 1, then this would indicate that v contributes a lot of value to the fractional solution. Therefore, it must have some importance and we want to select it with high probability. On the other hand, if a facility has a y_v value that is near 0, then this would imply that it does not contribute much to the fractional solution. Therefore it is acceptable to not open this facility in the integral solution.

The main result of this section is that this simple algorithm leads to a constant factor approximation.

THEOREM 1. *There is an LP-rounding 0.19-approximation for maximum facility location.*

The proof of this Theorem relies on a few auxiliary lemmas. First, we argue that the solution T output is feasible. Then we argue that the expected value of the set R is at least a constant factor of the value of the fractional solution (x, y). Finally, we show that the expected value of T is at least a constant factor of the expected value of R.

LEMMA 2. *The set T returned by* SELECT-AND-FILTER *is independent.*

PROOF. Suppose, for the sake of contradiction, that T is not feasible. This means there are in T two facilities v and v' whose intervals intersect. Without loss of generality assume that the start point of I_v is to the right of the start point of $I_{v'}$. Since $v' \in T$, it follows that $v' \in R$. But then v should not belong in T since the start point of I_v lies in $I_{v'}$, a contradiction. □

LEMMA 3. *Let R be the set sampled by* SELECT-AND-FILTER*, then*

$$\mathbb{E}[f(R)] \geq (1 - e^{-\alpha}) \sum_{u \in C, v \in F} w_{uv}\, x_{uv}.$$

PROOF. We focus on a fixed but arbitrary client $u \in C$. Let us rename the facilities $v_1, v_2, \ldots, v_k$ so that $w_{uv_1} \leq w_{uv_2} \leq \ldots \leq w_{uv_k}$, and define $w_{uv_0} = 0$. Finally, set $w(u) = \max_{v \in R} w_{uv}$.

We aim to show that $\mathbb{E}[w(u)]$, the contribution of u to $\mathbb{E}[f(R)]$, is at least a constant times $\sum_{v \in F} w_{uv}\, x_{uv}$, the contribution of u to the value of (x, y).

Let $\hat{Y}_v$ be an indicator variable that is 1 if $v \in R$ and 0 otherwise, then

$$\begin{aligned}
\Pr[w(u) < w_{uv_i}] &= \Pr\left[\bigwedge_{j=i}^{k} \hat{Y}_{v_j} = 0\right] \\
&= \prod_{j=i}^{k} \Pr[\hat{Y}_{v_j} = 0] = \prod_{j=i}^{k} (1 - \alpha y_{v_j}) \\
&\leq \left(1 - \frac{\alpha \sum_{j=i}^{k} x_{uv_j}}{k - i + 1}\right)^{k-i+1},
\end{aligned}$$

where the last inequality follows from the arithmetic-mean geometric-mean inequality and the feasibility of (x, y).

Let $\tilde{x} = \sum_{j=i}^{k} x_{uv_j}$ and $r = k - i + 1$. We observe that the function $a(\tilde{x}) = 1 - \left(1 - \frac{\alpha \tilde{x}}{r}\right)^r$ is concave on the interval $\tilde{x} \in [0, 1]$, and thus can be lowerbounded by the linear function $b(\tilde{x}) = \left[1 - \left(1 - \frac{\alpha}{r}\right)^r\right] \tilde{x}$. This is because $a(0) = b(0)$ and $a(1) = b(1)$. Therefore,

$$\begin{aligned}
\Pr[w(u) \geq w_{uv_i}] &= 1 - \Pr[w(u) < w_{uv_i}] \\
&\geq 1 - \left(1 - \frac{\alpha \sum_{j=i}^{k} x_{uv_j}}{k - i + 1}\right)^{k-i+1} \\
&\geq \left[1 - \left(1 - \frac{\alpha}{k - i + 1}\right)^{k-i+1}\right] \sum_{j=i}^{k} x_{uv_j} \\
&\geq (1 - e^{-\alpha}) \sum_{j=i}^{k} x_{uv_j}.
\end{aligned}$$

We now express $\mathbb{E}[w(u)]$ in terms of $\Pr[w(u) \geq w_{uv_i}]$

$$\begin{aligned}
\mathbb{E}[w(u)] &= \sum_{j=1}^{k} w_{uv_j} \Pr[w(u) = w_{uv_j}] \\
&= \sum_{j=1}^{k} \left(\sum_{i=1}^{j} w_{uv_i} - w_{uv_{i-1}} \right) \Pr[w(u) = w_{uv_j}] \\
&= \sum_{i=1}^{k} \sum_{j=i}^{k} (w_{uv_i} - w_{uv_{i-1}}) \Pr[w(u) = w_{uv_j}] \\
&= \sum_{i=1}^{k} (w_{uv_i} - w_{uv_{i-1}}) \Pr[w(u) \geq w_{uv_i}]
\end{aligned}$$

Now we can use our bound on $\Pr[w(u) \geq w_{uv_i}]$ to lowerbound the expected value of $w(u)$

$$\begin{aligned}
\mathbb{E}[w(u)] &= \sum_{i=1}^{k} (w_{uv_i} - w_{uv_{i-1}}) \Pr[w(u) \geq w_{uv_i}] \\
&\geq (1 - e^{-\alpha}) \sum_{i=1}^{k} (w_{uv_i} - w_{uv_{i-1}}) \sum_{j=i}^{k} x_{uv_j} \\
&= (1 - e^{-\alpha}) \left(\sum_{i=1}^{k} \sum_{j=i}^{k} w_{uv_j} x_{uv_j} - \sum_{i=1}^{k} \sum_{j=i}^{k} w_{uv_{i-1}} x_{uv_j} \right) \\
&= (1 - e^{-\alpha}) \sum_{j=1}^{k} x_{uv_j} \sum_{i=1}^{j} (w_{uv_i} - w_{uv_{i-1}}) = (1 - e^{-\alpha}) \sum_{j=1}^{k} x_{uv_j} w_{uv_j}.
\end{aligned}$$

Finally, by linearity of expectation, we have

$$\mathbb{E}[f(R)] = \sum_{u \in C} \mathbb{E}[w(u)] \geq (1 - e^{-\alpha}) \sum_{u \in C, v \in F} x_{uv} w_{uv},$$

which is precisely what we needed to prove. □

LEMMA 4. *Let R and T be the sets computed in* SELECT-AND-FILTER *then*

$$\mathbb{E}[f(T)] \geq (1 - \alpha)\mathbb{E}[f(R)].$$

PROOF. Let $v \in F$ and p be the start point of I_v. Let us start by estimating the probability that $v \notin T$ given that $v \in R$. Recall that v is removed from R only if there exists $v' \in R$ such

that $p \in I_{v'}$. Therefore,

$$\Pr[v \notin T \mid v \in R] = \Pr\left[\bigvee_{\substack{v' \in F-v: \\ p \in I_{v'}}} v' \in R\right] \leq \sum_{\substack{v' \in F-v: \\ p \in I_{v'}}} \Pr[v' \in R] = \sum_{\substack{v' \in F-v: \\ p \in I_{v'}}} \alpha y_{v'} \leq \alpha,$$

where the first inequality follows from union bound, and the second from the feasibility of (x, y). Therefore,

$$\Pr[v \in T \mid v \in R] \geq (1 - \alpha). \tag{4.1}$$

Vondrák et al. (2011) in Theorem 1.3 showed[1] that (4.1) implies that

$$\mathbb{E}[f(I)] \geq (1 - \alpha)\mathbb{E}[f(R)],$$

when f is monotone submodular. From Lemma 1 we know that f is indeed monotone submodular. □

PROOF. (of Theorem 1) Let R and T be the sets chosen by SELECT-AND-FILTER. By Lemma 2 the set T is independent. Its expected value is

$$\mathbb{E}[f(T)] \geq (1 - \alpha)\mathbb{E}[f(R)] \geq (1 - \alpha)(1 - e^{-\alpha}) \sum_{u \in C, v \in F} w_{uv} x_{uv},$$

where the first inequality follows from Lemma 4 and the second inequality from Lemma 3.

The best approximation ratio is attained at $\alpha = 0.44$, where we get 0.199. □

4.1.1 Algorithm Engineering

We briefly discuss two ideas for improving the observed performance of SELECT-AND-FILTER without sacrificing its theoretical guarantees: Picking the optimal α and resolve conflicts less aggressively.

First, notice that our choice of α in the proof of Theorem 1 may not be the best for a particular instance. Instead of picking a single value of α we pick the "best" α as follows. For each $v \in F$,

[1] Although the proof cited uses a different set R no assumption is used in the proof other than R is a random variable.

we generate a random number $r_v \in [0, 1)$. In SELECT-AND-FILTER, we can think of a facility v being added to the set R if $r_v \leq \alpha y_v$. Therefore there are discrete values of α for which we add a new facility v to our set R. Let $R_\alpha = \{ v \in F \mid r_v \leq \alpha y_v \}$ and $\alpha_1 < \alpha_2 < \cdots < \alpha_n$ be the interesting values of α when R_α changes; that is, $\alpha_j = r_v / y_v$ for some $v \in F$. Observe that $R_{\alpha_1} \subset R_{\alpha_2} \subset \cdots \subset R_{\alpha_n}$. Then we can simply compute T_{α_i} for each value of α_i and return the set maximising $f(T_{\alpha_i})$.

Second, recall that the algorithm resolves conflicts of R by keeping only those facilities whose start point is not contained in another interval in R. Consider a maximal subset $A \subseteq R$ such that the union of their intervals forms a single interval. For each such A, the filtering step of SELECT-AND-FILTER would only place the facility in A with the leftmost start point into T. Instead, we can be less aggressive and pick a maximal independent subset of A that includes the leftmost interval. Thus, we are guaranteed to pick an independent set that is a superset of the set chosen by the simple version. However, just because we choose a superset, does not mean we have designed an algorithm with a better approximation ratio. Consider the interval graph in Figure 4.1. In both the original and modified algorithm, both methods are only able to select the left-most interval, as it overlaps with every other interval.

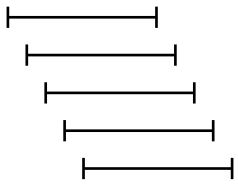


FIGURE 4.1: An instance where both Feldman's contention resolution scheme (Feldman, 2013) and our modified scheme can only select a single interval. In this case only the left-most interval will be selected.

4.2 Dependent Rounding

Our second algorithm uses a more refined rounding routine. Instead of sampling and removing intervals that have conflicts, we gradually shift the fractional solution to an integral using a dependent rounding (Gandhi et al., 2006) routine.

Before we can describe the algorithm we need a few definitions. Given a solution (x, y) to (LP), let $S = \{ v \in F \mid 0 < y_v < 1 \}$ be the set of facilities with a non-integral value.

DEFINITION 4. For a point $p \in P$, let slack(p) denote the slack of its corresponding constraint in (LP)

$$\text{slack}(p) = 1 - \sum_{v \in F : p \in I_v} y_v.$$

DEFINITION 5. A point $p \in P$ is *tight* if slack$(p) = 0$.

Algorithm 2 Dependent rounding.

function DEPENDENT-ROUNDING
  $(x, y) \leftarrow$ solve (LP)
  while $\exists$ non-integral facility **do**
    $S \leftarrow \{ v \in F \mid 0 < y_v < 1 \}$
    $M_1, M_2 \leftarrow$ SELECT-SUBSETS(S, y)
    For each $i = 1, 2$ set $P_i \leftarrow$ points that intersect an interval in M_i
    $\varepsilon \leftarrow \min\left(\min_{p \in P_1 \setminus P_2} \text{slack}(p), \min_{v \in M_2} y_v\right)$
    $\delta \leftarrow \min\left(\min_{p \in P_2 \setminus P_1} \text{slack}(p), \min_{v \in M_1} y_v\right)$
    For each $v \in F$ set $y'_v = \begin{cases} y_v - \delta, v \in M_1, \\ y_v + \delta, v \in M_2, \\ y_v, \quad \text{otherwise.} \end{cases}$
    For each $v \in F$ set $y''_v = \begin{cases} y_v + \varepsilon, v \in M_1, \\ y_v - \varepsilon, v \in M_2, \\ y_v, \quad \text{otherwise.} \end{cases}$
    $y = \begin{cases} y' & \text{with probability } \frac{\varepsilon}{\varepsilon+\delta} \\ y'' & \text{with probability } \frac{\delta}{\varepsilon+\delta} \end{cases}$
  end while
  return $\{ v \in F \mid y_v = 1 \}$
end function

Our algorithm starts by finding an optimal fractional solution (x, y) to the linear relaxation (LP). Then we iteratively modify this solution as follows. First, we select two disjoint independent subsets $M_1, M_2 \subset S$. Then, we randomly update the y-values of facilities in $M_1 \cup M_2$ so that at least one more facility is integral or at least one more point is tight. This is repeated until there are no more fractionally opened facilities.

Let us describe in more detail how the update is carried out. For $i = 1, 2$, let $P_i \subseteq P$ be those points that intersect an interval in M_i. We compute the following two quantities

$$\varepsilon = \min\left(\min_{p \in P_1 \setminus P_2} \text{slack}(p), \min_{v \in M_2} y_v\right),$$

$$\delta = \min\left(\min_{p \in P_2 \setminus P_1} \text{slack}(p), \min_{v \in M_1} y_v\right).$$

Finally, with probability $\frac{\varepsilon}{\varepsilon+\delta}$ we replace the current fractional solution y with a new solution

$$y'_v = \begin{cases} y_v - \delta, v \in M_1, \\ y_v + \delta, v \in M_2, \\ y_v, \quad \text{otherwise.} \end{cases}$$

Otherwise, with complementary probability $\frac{\delta}{\varepsilon+\delta}$, we replace y with the new solution

$$y''_v = \begin{cases} y_v + \varepsilon, v \in M_1, \\ y_v - \varepsilon, v \in M_2, \\ y_v, \quad \text{otherwise.} \end{cases}$$

The pseudocode of the main routine is given in Algorithm 2 while the pseudocode for picking M_1 and M_2 appears in Algorithm 3.

Algorithm 3 Creating subsets M_1 and M_2.

```
function SELECT-SUBSETS(S, y)
    P̃ ← the set of tight points in S sorted from left to right
    v_1 ← a facility in S whose interval the has leftmost start point
    v_2 ← a facility in S − v_1 whose interval starts at the first tight point in P̃
    M_1 ← { v_1 }
    M_2 ← { v_2 }
    for every tight point t in P̃ (in sorted order) do
        if I_{v_1} contains t and I_{v_2} contains t then
            continue
        end if
        v ← the facility in S whose interval starts at t
        if I_{v_2} contains t then
            M_1 ← M_1 + v
            v_1 ← v
        else if I_{v_1} contains t then
            M_2 ← M_2 + v
            v_2 ← v
        else
            break
        end if
    end for
    return M_1, M_2
end function
```

The following Theorem is the main result of this section.

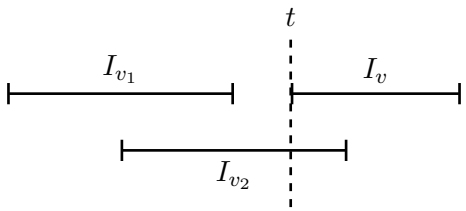


FIGURE 4.2: The case when only one of the intervals stabs the next tight point. Let I_v be the interval starting at point t. If t only intersects I_{v_2}, then since we scan from left to right, I_v cannot possibly intersect I_{v_1}.

THEOREM 2. *Let T be the set returned by* DEPENDENT-ROUNDING. *Then T is independent and $\Pr[v \in T] = y_v$ for all $v \in F$.*

We break the analysis into a series of lemmas. First, we prove some properties of the output of SELECT-SUBSETS. Then, we prove that the expected value of the fractional solution at the end of an iteration equals its value at the beginning of the iteration. Finally, we show that in each iteration we either gain an additional integral facility or a new tight point.

LEMMA 5. *Let M_1 and M_2 be the sets computed by* SELECT-SUBSETS. *Then M_1 and M_2 are disjoint and both are independent. Furthermore, if $p \in P$ is a tight point then either p stabs both M_1 and M_2, or none of them.*

PROOF. Observe that every tight point must occur at an interval start point. Let v_1 be the facility whose interval I_v is the first in S (from left to right). If the first tight point is to the right of the start point of I_{v_1} then I_{v_2} is well-defined. Otherwise, if the start point of I_{v_1} is tight there must be another facility starting at that point (since S is the set of non-integral facilities) so I_{v_2} is well-defined in this case too.

Now that we have shown the invariant holds after initialisation, we aim to prove that M_1 and M_2 are independent. To do this, we need to show that the loop invariant also holds at the end of each iteration. Consider the next tight point t and facility v starting at t. Suppose t stabs I_{v_2} only but it does not stab I_{v_1}. Since we scan from left to right, I_{v_1} is to the left of t and I_v starts at t, as shown in Figure 4.2. Therefore we can add v to M_1 correctly and update $v_1 = v$. The case when t stabs I_{v_1} but not I_{v_2} is handled in a similar fashion, but adding v to M_2 and updating $v_2 = v$. Finally, if t stabs both I_{v_1} and I_{v_2}, we continue to the next tight point; otherwise, if t

stabs neither I_{v_1} or I_{v_2}, the algorithm stops. Clearly there are a finite number of tight points, therefore the loop will eventually terminate. Thus the invariant holds throughout.

Now consider a tight point $p \in P$. If p was never considered by the for loop then p is strictly to the right or to the left of every interval in $M_1 \cup M_2$, and thus does not stab any of them. Otherwise, in the iteration when p was considered by the algorithm it either intersected both I_{v_1} and I_{v_2}, in which case we are done, or it added v so that the property holds. □

LEMMA 6. *Consider an iteration of* DEPENDENT-ROUNDING. *Let y be the fractional solution at the beginning, and $\hat{y}$ be the solution at the end of the iteration. Then for every $v \in F$ we have $\mathbb{E}[\hat{y}_v] = y_v$. Furthermore, $\hat{y}$ obeys the independence and non-negativity constraints of* (LP).

PROOF. If we have a facility $v \in F$ that is not in M_1 or M_2, then it is clear that the equality holds as its value does not change. For each facility $v \in M_1$ we have

$$\begin{aligned}\mathbb{E}[\hat{y}_v] &= \left(\frac{\varepsilon}{\varepsilon+\delta}\right)(y_v - \delta) + \left(\frac{\delta}{\varepsilon+\delta}\right)(y_v + \varepsilon) \\ &= \frac{\varepsilon y_v - \varepsilon\delta + \delta y_v + \varepsilon\delta}{\varepsilon+\delta} \\ &= y_v.\end{aligned}$$

A similar argument applies for facilities $v \in M_2$.

Notice that $\hat{y}_v \geq 0$ by our definition of ε and δ since each of these values is less than or equal to the minimum y-value of the subset of variables that are decreased in each case.

Finally, let us now argue that y obeys the independence constraints of (LP). Let $p \in P$ be an arbitrary point. If $p \in P_1 \cap P_2$ then its constraint is obeyed since the increase/decrease of a facility in M_1 that it stabs is compensated with a corresponding decrease/increase of a facility in M_2 that it stabs. If $p \in P_1 \setminus P_2$ then by the definition of ε the independence constraint at p is obeyed. Similarly, if $p \in P_2 \setminus P_1$ then by the definition of δ the independence constraint at p is obeyed. Finally, $p \notin P_1 \cup P_2$ then the slack of p does not change so the independence constraint at p is obeyed as well. □

LEMMA 7. *In each iteration of* DEPENDENT-ROUNDING *we either gain a new integral facility or a new tight point.*

PROOF. Let us argue that $\varepsilon > 0$ and $\delta > 0$. Note that $\min_{v \in M_2} y_v > 0$ since all facilities in M_2 are non-integral, and that $\min_{p \in P_1 \setminus P_2} \text{slack}(p) > 0$ since all tight points that stab M_1 also stab M_2. Putting these two facts together, we get $\varepsilon > 0$. A similar line of reasoning yields $\delta > 0$. Therefore, no matter what type of update we perform, we always move to a different fractional solution.

Suppose that the algorithm replaces y with y'. Furthermore, assume that $\delta = y_v$ for some $v \in M_1$. It follows that $y'_v = 0$, so we gain a new integral facility. Similarly, assume that $\delta = \text{slack}(p)$ for some $p \in P_2 \setminus P_1$. After the update the slack at p will be 0, so we gain a new tight point since by Lemma 5 the point p cannot be tight. Therefore, either y' has one additional integral facility, or one additional tight point. A similar argument shows that the same holds if the algorithm decides to replace y with y''. □

PROOF. (Of Theorem 2) By Lemma 7, in each iteration we either gain one additional integral facility or one additional tight point. Therefore, after $2|F|$ iterations the algorithm must terminate with an integral solution $\hat{y}$.

By Lemma 6, in each iteration we maintain feasibility. It follows that the set $T = \{ v \in F \mid \hat{y}_v = 1 \}$ is feasible. Finally, we show that $\Pr[v \in T] = y_v$ for all $v \in F$. Let y_v^i be the value of a facility $v \in F$ at the end of iteration i of the algorithm. We define $y_v^0 = y_v$ and $y_v^{2|F|} = \hat{y}_v$. From Lemma 6, we know that $\forall i \geq 0 : \mathbb{E}[y_v^{i+1}] = y_v^i$. Let $v \in F$ be a facility in M_1 in the last iteration. At this point, we can either increase the value of v by some ε such that $\hat{y}_v = 1$ or decrease the value by δ such that $\hat{y}_v = 0$. Therefore

$$\begin{aligned} y_v = y_v^0 &= \mathbb{E}[y_v^1] = \ldots = \mathbb{E}[\hat{y}_v] \\ &= \Pr[\hat{y}_v = 1](y_v^{2|F|-1} + \varepsilon) + \Pr[\hat{y}_v = 0](y_v^{2|F|-1} - \delta) \\ &= \Pr[\hat{y}_v = 1] = \Pr[v \in T]. \end{aligned}$$

This property also holds for any facility $v \in M_2$. If a facility $v \in F$ was assigned an integral value by (LP), then it is easy to see that this property also holds. □

In terms of the differences between these two algorithms discussed, they both exhibit a trade-off in solution quality versus time. The first algorithm (independent rounding) presented has a theoretical bound, however will often be slower than the second, as for large instances we need to evaluate the objective function in (3.1) many times. On the other hand, the runtime of

the second algorithm (dependent rounding) is dependent on the number of tight points in the instance. Therefore for instances with a small number of tight points, the dependent algorithm will most likely have a faster running time.

We close this section by noting that while the output set T is independent and $\Pr[v \in T] = y_v$, we cannot prove the guarantee of Lemma 3 for $\mathbb{E}[f(T)]$ since the rounding decisions are not independent of one another. However, we can show that this algorithm admits a non-constant approximation.

THEOREM 3. *The algorithm* DEPENDENT-ROUNDING *admits a non-constant approximation.*

PROOF. For some integer k, consider the following interval graph in Figure 4.3.

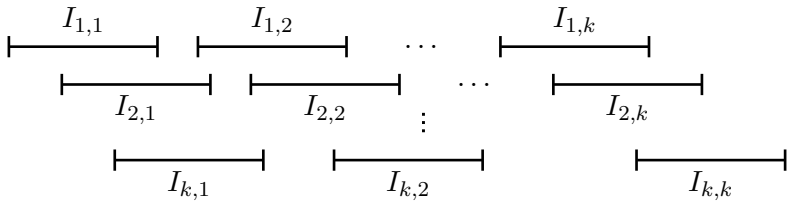


FIGURE 4.3: An instance that leads to a non-constant approximation. The DEPENDENT-ROUNDING algorithm will only ever open one row of intervals. The optimal solution is to open the intervals $I_{1,1}, I_{2,2}, \ldots, I_{k,k}$ across the diagonal.

Define $U_i = \{ I_{i,j} \mid j \in [1,k] \}$ for some $i \in [1,k]$. Create k clients $u_1, u_2, \ldots u_k$. For each client u_i, assign it to every facility that corresponds to the interval in the set U_i with weight 1. This means to satisfy a client u_i, we only need to open one facility whose interval is in the set U_i.

The optimal solution is to open the facility corresponding to interval $I_{1,1}$, then $I_{2,2}$ and so on until $I_{k,k}$. Therefore the value of the optimal solution is k. On the other hand, an optimal fractional solution to this problem is to assign every facility a value of $1/k$.

Now consider the first iteration of DEPENDENT-ROUNDING. It is easy to see that $M_1 = U_1$ and $M_2 = U_2$. When the rounding occurs, we will either shift all of the facilities in M_1 up to a value of $2/k$ or vice versa. This process is continued until every facility in U_j has a value of 1 (for some $1 \leq j \leq k$), and every facility in every other row U_i has value 0 for $i \neq j$. Since we only opened a single row, we can only connect a single client u_j, for a cost of 1. Therefore this instance results in a $1/k$-approximation, which is non-constant. □

4.3 Hardness

We conclude this chapter by proving an inapproximaibiltiy result for the maximum facility location problem, by reducing from the the maximum coverage problem.

4.3.1 Reducing from Maximum Coverage

In the maximum coverage problem we are given an integer k, a universe $\mathcal{U} = \{ u_1, u_2, \ldots, u_n \}$, and a collection of subsets $\mathcal{S} = \{ S_1, S_2, \ldots, S_m \}$ such that $\forall S \in \mathcal{S} : S \subseteq \mathcal{U}$. Our goal is to select a collection $\mathcal{C} \subseteq \mathcal{S}$ of these subsets, where $|\mathcal{C}| \leq k$ and we maximise the quantity

$$\left| \bigcup_{S_i \in \mathcal{C}} S_i \right|.$$

A simple greedy algorithm is to pick a set S at each iteration (while $|\mathcal{C}| < k$) maximising $|S \setminus \cup_{T \in \mathcal{C}} T|$. This method results in a $\left(1 - \frac{1}{e}\right)$-approximation algorithm. Furthermore, we know that there is no polynomial algorithm for this problem with a ρ-approximation for $\rho > \left(1 - \frac{1}{e}\right)$ unless P = NP (Feige, 1998; Khuller et al., 1999).

To begin the reduction to maximum facility location, we create a client c_u for each element $u \in \mathcal{U}$. The main constraint we model is that we can only pick at most k subsets from our collection $\mathcal{S}$. Therefore we create km facilities. For each subset $S \in \mathcal{S}$, we create the facilities $\phi_{S,1}, \phi_{S,2}, \ldots, \phi_{S,k}$ representing the subset S. From these facilities, we form k cliques in the interval graph such that for every value of $i \in [1, k]$, every pair of intervals in the set $\{ I_{\phi_{S,i}} \mid S \in \mathcal{S} \}$ is overlapping. This ensures that we can only choose to open one facility (i.e. select one subset from the collection) from each clique, and this can be done at most k times.

If an element $u \in \mathcal{U}$ is in a set $S \in \mathcal{S}$, then we add the edges

$$(c_u, \phi_{S,1}), (c_u, \phi_{S,2}), \ldots, (c_u, \phi_{S,k})$$

all with weight 1. See Figure 4.4. This means that once we choose to open the facility $\phi_{S,i}$, we are essentially covering all of the elements $u' \in S$, by assigning all of the clients $c_{u'}$ to $\phi_{S,i}$.

LEMMA 8. *In the reduction above, the value of the maximum facility location instance equals the value of the maximum coverage instance.*

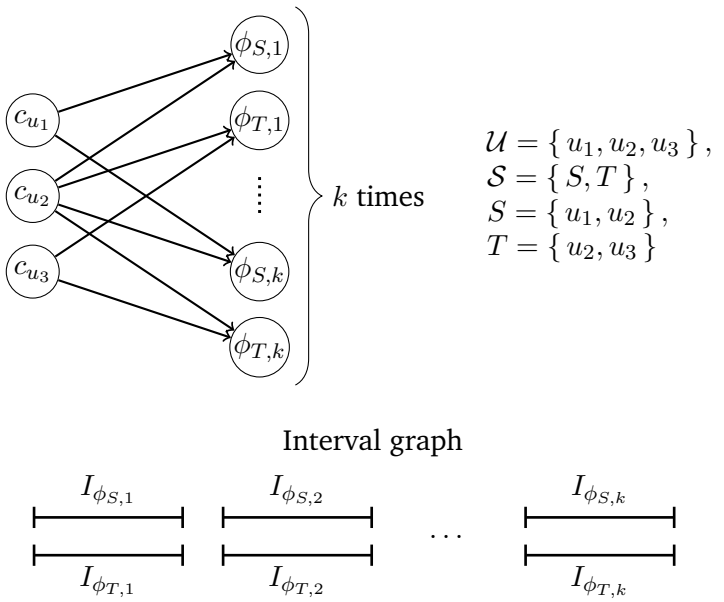


FIGURE 4.4: An example reduction from the maximum coverage problem to the maximum facility location problem. In this instance, there are three elements (u_1, u_2 and u_3) and two subsets (S and T). Given an integer k, create k intervals for each subset and create k cliques. In each clique, every pair of intervals overlap. This means we can select at most k subsets.

PROOF. Let Φ be the set of facilities opened and $\mathcal{C}_\Phi$ be the corresponding subsets we choose from the collection $\mathcal{S}$. Since we can pick at most one facility from each clique, and there are k of them, it follows that $|\mathcal{C}_\Phi| \leq k$.

In order to argue that the values are the same, we observe that by opening a facility $\phi_{S,i}$, we allow all of the clients connected to $\phi_{S,i}$ to be assigned. Since all of the edge weights are 1, this corresponds to how many elements we cover if we select the subset S. The second observation is that the instance cannot benefit by opening the facility $\phi_{S,i}$ and $\phi_{S,j}$ for $j \neq i$, as a client can only be assigned to a single open facility. This directly corresponds to covering no new elements. □

THEOREM 4. *For any $\varepsilon > 0$, there is no $(1 - 1/e + \varepsilon)$-approximation algorithm for maximum facility location, unless* P = NP.

PROOF. The result follows from Lemma 8 and the inapproximability result (Feige, 1998; Khuller et al., 1999), for maximum coverage. □

CHAPTER 5

Conflicting Predictions from Multi-mapping Reads

In this chapter we present an application of the maximum facility location problem. Ultra-high-throughput next-generation sequencing (NGS) technology allows researchers to rapidly sequence millions of DNA molecules, at a decreasing cost. These NGS instruments generate a huge number of short DNA sequences, known as 'reads'. If a high-quality reference genome is available for the sequenced organism, then one may try to map the reads from the donor genome back to their origin in the reference genome.

While this task sounds simple, it is in fact obfuscated by errors in sequencing, differences between the reference and donor genome, and other factors that may increase the difficulty of determining a reads origin in the reference genome. Typically, this means that a read may map to multiple (plausible) places in the reference gnome.

In this chapter, we will demonstrate how our work can be applied to this scenario. More precisely, our aim is to model the prediction of deletions from paired-end reads. These paired-end reads are obtained by sequencing both ends of DNA fragments of a donor genome that are 'roughly' of a certain size. A mapping of these two reads to the reference genome at a distance that is larger than what is expected indicates a potential deletion in the genome. The confidence in a mapping of a read is expressed by an alignment score that captures characteristics such as mismatches, and the likelihood of the fragment length according to the estimated fragment length distributions. See Figure 5.1.

Essentially, our goal is to assign these multi-mapping reads to high-quality alignments then in turn implies a set of conflict-free predictions. However, in order to apply the work thus far, we make the assumption that these predictions are represented by genomic intervals. This means that two predictions are in conflict if (and only if) their corresponding intervals overlap.

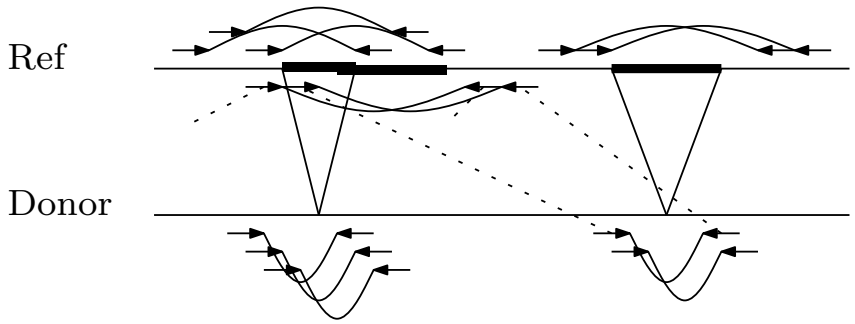


FIGURE 5.1: Alignments supporting conflicting deletions. Two reads (arrow heads) are connected by a curved line which represents a paired-end read. The two reads map to the reference genome (solid lines) at a distance larger than the distance between the sequenced ends (reads) of the donor fragments, indicating a potential deletion. Alternative wrong alignments (dotted lines) support an incorrect deletion that overlaps the correct deletion to the left and thus the two deletions cannot occur simultaneously on the same allele.

In the following sections, we first discuss how we model this problem as the maximum facility location problem. Afterwards, we discuss the implementation of the two algorithms discussed in Chapter 4. In Section 5.4, we run our algorithms on two main data sets in order to show the effectiveness of our algorithms and discuss the results.

5.1 Modelling the Problem

In this application, one can see that facilities correspond to candidate deletions (intervals) in the reference genome, while clients correspond to paired-end reads in the donor genome. A client (read) can be assigned to the facilities (deletions) that are supported by one of its edges (mappings). Assigning a client to a facility contributes the corresponding read alignment score to the overall weight. Therefore, starting from an initial set of candidate deletions, the optimal solution aims to find an independent set of facilities that minimise the number of false-positive predictions while maximising the number of true position deletions.

One can see that this problem is exactly the maximum facility location problem. Thus, we can apply the algorithms and techniques discussed here to this scenario. See Figure 5.2.

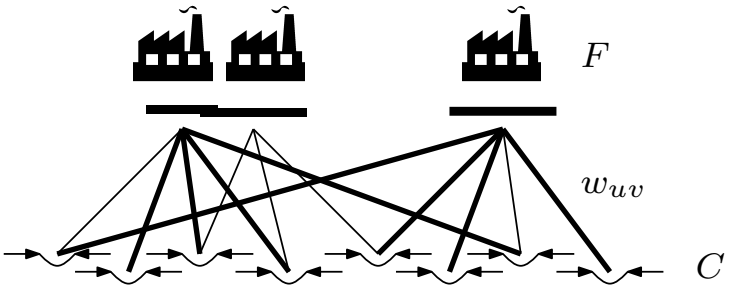


FIGURE 5.2: Clients C and facilities F represent paired-end reads and candidate deletions, respectively. Each facility is associated with an interval whose endpoints correspond to the breakpoints of the candidate deletion. Edges between clients and facilities indicate alignments supporting the corresponding deletion. Our confidence in the alignment is captured by a score w. A feasible solution (bold edges) to the *maximum facility location problem* assigns reads to an independent set of facilities.

5.2 Implementation

The implementation for this project was done in Java 8. The solver Gurobi (Gurobi, 2015) was used to solve the integer and linear programs. Essentially, the implementation is broken into two major components.

(1) The algorithms package. This package contains the implementation of all algorithms. These include the independent rounding algorithm (`optimalResolveContentions`, Section 4.1), the dependent rounding algorithm (`alternateRoundingScheme`, Section 4.2) and an additional implementation of the independent rounding algorithm that uses Feldman's conflict resolutions scheme (`optimalResolveContentionsFeldman`) (Feldman, 2013). Two additional greedy algorithms (`simpleGreedy`, Section 5.3) were also implemented to compare these current algorithms to our new methods. As a baseline, the integer program (IP) was also implemented (`solveInstance`).

(2) The utilities package. Classes such as the `Facility` class (containing information such as it's name, and the interval's start and end point), the `Client` class (containing it's name, and the cost of assigning a client to a facility) and an `Instance` class (which stores an array of both the facilities and clients in the instance).

See Figure 5.3 for a more precise breakdown. In the class diagram, each class has it's own set of useful functions. For example, the `Facility` class has the function `intersectsFacility` to determine if a two facilities v and v' have their corresponding intervals overlap. Two other useful

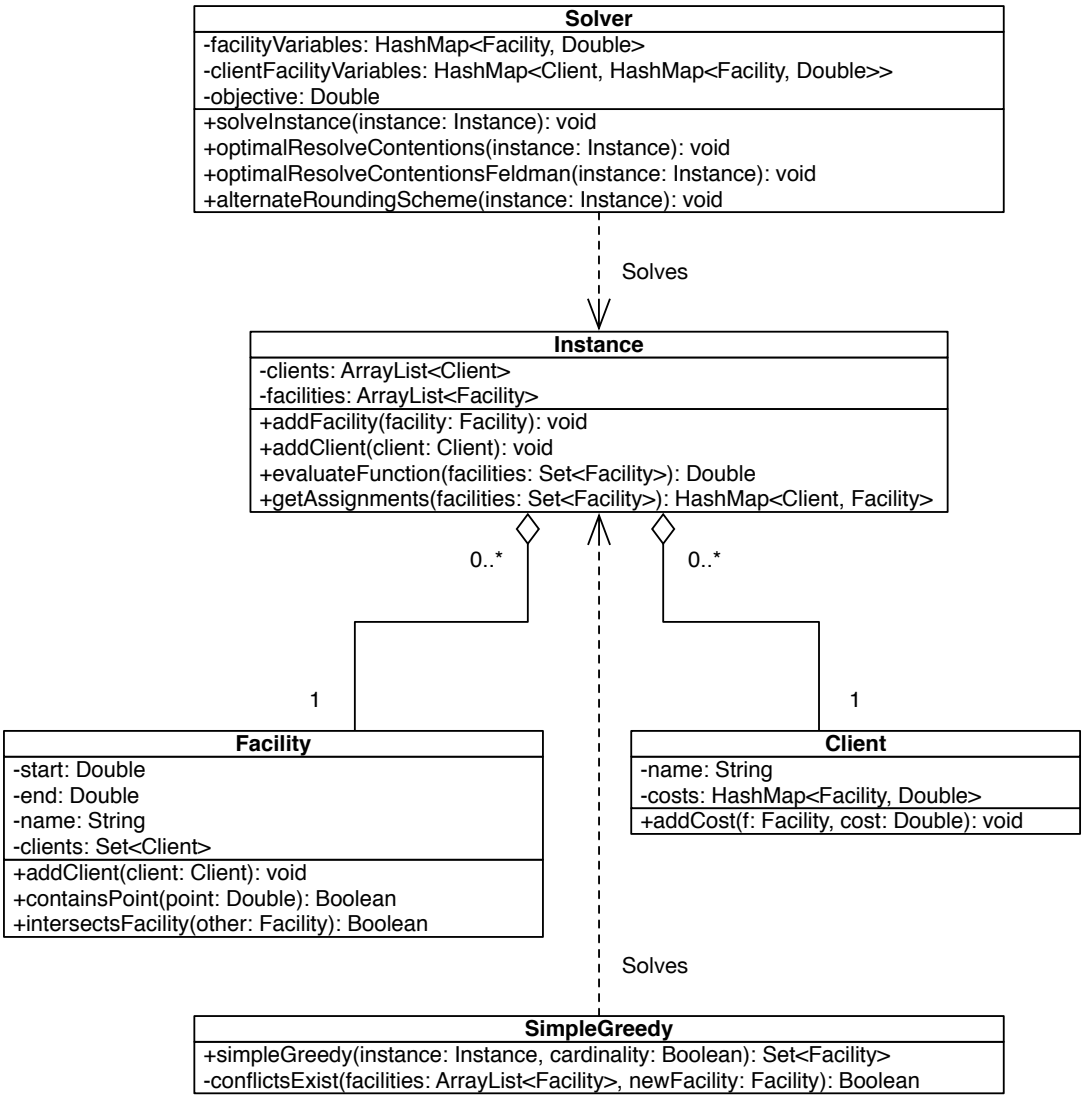


FIGURE 5.3: A simplified class diagram of the implementation. An `Instance` contains an array of facilities and clients. Each `Facility` object has an ID (`name`), start and end point of the associated interval, and the list of clients that are connected to the facility. A `Client` object has an ID (`name`), and a hash map (v, w) where v is the facility object and w is the cost of assigning the client to facility v. The two classes `Solver` and `SimpleGreedy` are responsible for solving instances of the problem.

functions are located in the `Instance` class. The first is `evaluateFunction`, which evaluates the objective function (3.1) given a subset of facilities. The `getAssignments` function returns the assignments of clients to facilities, given a subset of non-overlapping facilities. In other words, given a set $S \subseteq F$, it returns the map $\sigma_S : C \to S$. More precisely, $\sigma_S : u \mapsto \arg\max_{v \in S} w_{uv}$.

5.3 Related Work

Before discussing the experiments that were conducted, we briefly discuss some related work in the field. The conflicts between structural variants (SV) were first studied by Hormozdiari et al. (2009). Originally, overlapping structural variants were heuristically filtered in a post-processing step. However, the authors then integrated their conflict resolution method in follow up work (Hormozdiari et al., 2010). Furthermore, they show that the resulting combinatorial problem is NP-hard and propose a heuristic greedy algorithm. The algorithm they propose, which we will call GREEDY, is described in Algorithm 4.

Algorithm 4 The greedy algorithm described by Hormozdiari et al. (2010).

function GREEDY(F, C, w, I)
  $T \leftarrow \varnothing$
  while $\exists v \in F : I_v$ does not conflict with any intervals in $\{ I_{v'} \mid v' \in T \}$ **do**
    $v^\star \leftarrow$ the facility in $F \backslash T$ that covers the most new clients
    $T \leftarrow T + v^\star$
  end while
  return T
end function

In addition to this, we also decided to implement a variation of the greedy algorithm, known as WEIGHTED-GREEDY. The key difference is that WEIGHTED-GREEDY takes into account the weights between clients and facilities, rather than simply selecting facilities which cover the most amount of clients.

Algorithm 5 The greedy algorithm described by Hormozdiari et al. (2010).

function WEIGHTED-GREEDY(F, C, w, I)
  $T \leftarrow \varnothing$
  while $\exists v \in F : I_v$ does not conflict with any intervals in $\{ I_{v'} \mid v' \in T \}$ **do**
    $v^\star \leftarrow$ the facility in $F \backslash T$ that maximises $f(S + v^\star) - f(S)$
    $T \leftarrow T + v^\star$
  end while
  return T
end function

Finally, Feldman (2013) provided a 0.25-approximation based on rounding a fractional solution of the multilinear relaxation for f (see Section 2.3.2). Unfortunately, as discussed in Chapter 2, the current best algorithms for finding good solutions are impractical due to their high-degree-polynomial running time.

5.4 Experiments

To evaluate both the existing algorithms formulated by Hormozdiari et al. (2010) and ours, we ran the algorithms on two data sets. All of the experiments were run on a quad-core Intel Core i7 processor running at 2.7 GHz with 16GB of RAM.

In order to measure the performance of the algorithms, we look at the predicted deletions versus actual deletions and compare the recall and precision. We will define a predicted deletion to be a true positive TP if it overlaps with a true deletion and differs in length by no more than 100 base pairs (bp). From there, the definitions of a false positive FP, false negative FN and true negative TN follow. We can then define the precision P and recall R,

$$P = \frac{TP}{TP + FP}, \text{ and } R = \frac{TP}{TP + FN}.$$

However, we can observe that by this definition, there could be one predicted deletion which covers multiple true deletions or vice versa. Since our aim is to build a one-to-one correspondence between predicted and actual deletions, we can build a bipartite graph $B = (V, U; E)$ where V is the set of predicted deletions, U is the set of true deletions and $(i, j) \in E$ if and only if deletions corresponding to $i \in V$ and $j \in U$ overlap and their lengths differ by no more than 100 bp. Therefore, we can compute a maximum (cardinality) matching M. This means the set of all true positive predictions corresponds to all $i \in V$, where $(i, j) \in M$ for some $j \in U$.

5.4.1 Simulated Reads: Craig Venter

The first data set we simulate our algorithms on was designed by Marschall et al. (2012), which is based on Craig Venter's genome. In their paper, the authors proposed a new tool known as CLEVER (clique-enumerating variant finder) which is able to determine statistically significant deletions. These deletions then form an instance of the maximum facility location problem, and can be fed into our algorithms. From these opened facilities, we can derive our predicted deletions in Venter's genome. Note that CLEVER is just one tool that discovers structural variants. However, there is no reason that any other SV discovery method such as VariationHunter (Hormozdiari et al., 2009), GASV (Sindi et al., 2009), BreakDancer (Chen et al., 2009), HYDRA (Quinlan et al., 2010), MoDIL (Lee et al., 2009), PINDEL (Ye et al., 2009) or SV-seq2 (Zhang et al., 2012) could be used here.

TABLE 5.1: Recall and precision achieved by the different algorithms when provided with the candidate predictions of CLEVER (Marschall et al., 2012) (CLEVER-cand) on the simulated Craig Venter data set. We report recall for deletions of different length in a-b bases (Rec. a-b) as well as overall recall and overall precision. The last column gives the total running time in minutes.

	Rec. 20-49	Rec. 50-99	Rec. 100-50kb	Recall	Precision	runtime
CLEVER-cand	66.80	79.47	68.49	68.91	9.93	N/A
Independent	62.37	75.30	66.29	65.02	40.63	15.8
Dependent	63.10	75.96	66.32	65.59	40.49	4.5
IP	63.46	75.96	66.26	65.80	40.26	10.4
Greedy	63.02	75.14	66.42	65.44	41.47	1840.0
wGreedy	62.86	74.97	66.29	65.29	41.42	4095.0

The results of this experiment is given in Table 5.1. The table shows the recall and precision of various algorithms. As discussed in the previous paragraph, each algorithm was provided with all significant deletions produced by CLEVER (Marschall et al., 2012) (CELEVER-cand). We ran this simulated data on the SELECT-AND-FILTER algorithm with the added alterations discussed in Section 4.1.1 (Independent), the DEPENDENT-ROUNDING algorithm (Dependent), the GREEDY algorithm (Greedy) and the WEIGHTED-GREEDY, algorithm (wGreedy). Since the instances were reasonably small, we also solved the integer program (IP) using Gurobi (Gurobi, 2015) for benchmarking purposes.

Similar to the evaluation by Marschall et al. (2012), we consider deletions of a particular range. Specifically, ranges of size 20-49 bp (8502 true deletions), 50-99 bp (1822 true deletions) and 100-50000 bp (2996 deletions). Note that since the precision P requires the calculation of the number of false positives, which relies on the overall truth set, we report only the global precision. Refer to Table 5.2 for the size of each instance.

All methods are able to reduce the number of false positive calls made by CLEVER (CLEVER-cand in Table 5.1) significantly, at only a small cost in recall. The very similar accuracy of the predictions obtained by the maximum facility location algorithms and the greedy algorithms is remarkable, since they are based on different mathematical models. While the latter seeks a minimal set of deletions (facilities) to explain all reads, the algorithms developed in this work rely exclusively on alignment scores and conflicts between deletions and do not impose any further assumptions.

TABLE 5.2: The number of clients, facilities and edges for each Chromosome (1 to Y) in Craig Venter's genome.

	Clients	Facilities	Edges		Clients	Facilities	Edges
1	27490	11790	117577	13	15213	6651	64483
2	29164	12136	120904	14	10337	4341	43081
3	21891	9265	91488	15	9717	3884	39924
4	25733	10584	102946	16	11611	5161	49446
5	21297	9077	90053	17	13193	5884	58176
6	22495	9626	90902	18	12442	5440	51497
7	22733	9741	94009	19	11625	5346	51337
8	19027	8020	77575	20	9767	4222	39438
9	13439	5556	51817	21	5894	2599	23426
10	20595	9271	89273	22	7079	3270	32364
11	18604	8038	77780	X	9362	4010	37557
12	17823	7619	73013	Y	424	169	1261

In terms of runtime performance, the most notable trend is that the greedy heuristics were two orders of magnitude slower than the LP rounding heuristics, taking days rather than minutes to find a solution. This is because the greedy algorithms need to evaluate the objective function (3.1) many times, which can be expensive for large instances. Compare this to our solutions which solve a linear program and evaluate the objective function a few times. Surprisingly, the IP solver showed a performance comparable to our heuristics. This is probably due to the fact that the IP approach involves a single call to the highly optimised Gurobi (Gurobi, 2015) software library written in C, while our implementation uses Java. Nevertheless, we expect our rounding algorithms to scale better on larger instances.

Placing aside the precision and recall of our algorithms, we can also look at the gap in value between our algorithms and the optimal solution (provided by the IP). This is useful because it allows us to evaluate our algorithms from a theoretical perspective. Let λ be the value of the optimal solution to (IP) and $S \subseteq F$ be some independent set of facilities produced by one of the algorithms. Then the gap in between the value of the optimal solution λ and the value realised by the set S is

$$\frac{|\lambda - f(S)|}{\lambda}.$$

The results for chromosomes 1 to Y are given in Table 5.3. It is clear from the data that the dependent rounding algorithm is the clear winner, producing solutions that are all only 0.5% below the optimal solution. One can see that (surprisingly) the value of all algorithms are

TABLE 5.3: The value of each algorithm, for each chromosome in the Craig Venter simulated data. The relative gap between the value of the optimal solution (IP) and each algorithm is given in brackets (as a percentage).

	IP	Independent		Dependent		Greedy		wGreedy	
1	24550	24438	(0.45)	24505	(0.18)	24026	(2.13)	24035	(2.10)
2	26039	25927	(0.43)	25962	(0.29)	25504	(2.05)	25442	(2.29)
3	19634	19527	(0.54)	19597	(0.19)	19262	(1.89)	19262	(1.89)
4	22861	22797	(0.28)	22837	(0.10)	22461	(1.75)	22442	(1.83)
5	18902	18851	(0.27)	18875	(0.14)	18500	(2.12)	18509	(2.08)
6	20239	20154	(0.42)	20196	(0.21)	19849	(1.93)	19866	(1.84)
7	20141	20015	(0.63)	20099	(0.21)	19725	(2.06)	19743	(1.98)
8	16989	16893	(0.56)	16921	(0.40)	16594	(2.32)	16656	(1.96)
9	11956	11926	(0.25)	11928	(0.23)	11764	(1.60)	11737	(1.83)
10	18350	18256	(0.51)	18325	(0.14)	17957	(2.14)	17958	(2.13)
11	16715	16617	(0.58)	16686	(0.17)	16325	(2.33)	16341	(2.23)
12	15914	15846	(0.42)	15879	(0.22)	15593	(2.01)	15597	(1.99)
13	13667	13562	(0.77)	13614	(0.39)	13372	(2.16)	13387	(2.05)
14	9308	9271	(0.40)	9297	(0.12)	9125	(1.96)	9141	(1.79)
15	8717	8669	(0.55)	8690	(0.31)	8552	(1.89)	8556	(1.85)
16	10283	10181	(1.00)	10257	(0.26)	10074	(2.04)	10062	(2.15)
17	11568	11518	(0.43)	11536	(0.28)	11325	(2.10)	11320	(2.14)
18	11105	11054	(0.46)	11086	(0.17)	10911	(1.74)	10914	(1.72)
19	10134	10056	(0.77)	10101	(0.33)	9933	(1.98)	9926	(2.05)
20	8755	8702	(0.61)	8723	(0.37)	8557	(2.27)	8561	(2.22)
21	5289	5265	(0.45)	5287	(0.04)	5169	(2.27)	5158	(2.47)
22	6261	6189	(1.15)	6249	(0.19)	6072	(3.02)	6082	(2.85)
X	8412	8393	(0.23)	8405	(0.08)	8260	(1.81)	8255	(1.87)
Y	369	369	(0.00)	369	(0.00)	365	(1.08)	366	(0.81)

quite high, producing nearly optimal solutions. To explain this, one can look at the instances themselves. A simple visualiser was implemented on top of the existing Java code base to look at the interval graph. From this, we observed that in all instances, the interval graph is extremely sparse with few cliques (sets of intervals that all overlap each other). On average, each clique contained about 2-3 intervals, and the average length of the clique was 40-60. Furthermore, the

average distance between each clique is about a gap of 40000-50000. In order to further aid the explanation, we performed further analysis on how the clients were assigned to these facilities. On average, a client was assigned to 3-4 facilities in a single clique 65% of the time. While the remaining 35% of the time, a client was assigned to roughly 5 facilities across multiple cliques (usually 2-3 cliques).

This leads us to the explanation that since there aren't many conflicting intervals, any simple method can solve the instances well. This primarily explains why the greedy method performs so well. In the instance where there's a large clique, many clients are usually assigned to many of the facilities in this clique, so picking any of their intervals will work. On the other hand, clients who are assigned to many facilities across a few cliques will also be satisfied as we can just pick a facility from any clique.

5.4.2 Illumina Reads: NA12878

In order to further demonstrate the utility of our algorithms in this application, we now turn to a real data set. We evaluate our methods on publicly available Illumina sequencing data of the NA12878 genome.[1] As before, we run CLEVER (Marschall et al., 2012) on this data and feed the statistically significant deletions predicted into our algorithms as the set of facilities. From facilities opened by the algorithms, we can derive the set of predicted deletions in NA12878's genome.

Since this data is not simulated, it is harder to look at the precision and recall since we don't have the exact ground truth (i.e. the set of true deletions). However, there are two truth sets currently available, created by Layer et al. (2014). The first truth set, known as *1000G* contains 3376 non-overlapping deletions validated in NA12878 by Mills et al. (2011). The second truth set, called *long-read*, contains 4095 deletions. Refer to Table 5.5 for the instance size of each chromosome in the NA12878 genome.

Recall and precision are defined by matching predicted and true deletions as described at the beginning of this section. However, for this real data set we do not know the complete set of true deletions. Therefore we applied the following precision correction to the output of all benchmarked methods. Let $\mathcal{D}$ be the set of predicted deletions (opened facilities). Predicted

[1] This genome is from the European Nucleotide Archive, ERA172924. See ebi.ac.uk/ena/data/view/ERA172924.

TABLE 5.4: Recall and precision with respect to truth sets *1000G* and *long-read* achieved by the different algorithms when provided with the candidate predictions of CLEVER (Marschall et al., 2012) (CLEVER-cand) on the Illumina data set for NA12878. We report recall for deletions of different length in a-b bases (Rec. a-b) as well as overall recall and overall precision. The number of true deletions in the three different size ranges are shown in brackets for the two truth sets. The last column gives the total running time in minutes.

	Rec. 20-49	Rec. 50-99	Rec. 100-50kb	Recall	Precision	runtime
1000G (47, 156, 2989)						
CLEVER-cand	5.4	35.2	65.8	63.5	8.8	N/A
IP	5.4	34.0	62.4	60.3	69.3	10.5
Dependent	5.4	34.0	62.3	60.1	69.03	3.1
Independent	5.4	34.0	62.3	60.2	69.11	7.0
Greedy	5.4	34.6	62.2	60.2	69.79	1737.5
wGreedy	5.4	34.6	62.4	60.3	69.99	4947.2
long-read (101, 522, 3443)						
CLEVER-cand	23.5	44.5	79.5	73.9	9.9	N/A
IP	18.8	38.4	72.6	67.1	78.9	10.5
Dependent	20.0	38.4	73.0	67.4	79.4	3.1
Independent	20.0	38.2	73.0	67.4	79.5	7.0
Greedy	20.0	37.8	72.6	67.1	80.1	1737.5
wGreedy	20.0	37.8	72.3	66.8	79.8	4947.2

deletions $\mathcal{D}$ that do not match any true deletion and that are not supported by at least one read that in turn supports at least one true deletion through one of its alternative alignments do not count as false positive hits. Let $d \in \mathcal{D}$ be one of these deletions. The rationale behind this approach is that none of the reads that map to d would contribute to a true positive hit. This is potentially due to true deletions in NA12878's genome that are missing in our truth set. Thus there is no meaningful way of comparing the performance of post-processing methods among predictions in $\mathcal{D}$. Furthermore, their inclusion would blur the results on the remaining predictions where the existence of an alternative assignment to a true deletion indicates a true false positive prediction.

Table 5.4 reports recall and precision of the different methods with respect to truth sets *1000G* and *long-read*. In terms of recall we consider the same size ranges as in the simulated Craig Venter benchmark. The number of (validated) true deletions for NA12878 in the relevant ranges of 20-49 and 50-99 are 47 and 156 for *1000G*. For *long-read*, the number of (validated) true deletions in the range 20-49 and 50-99 are 101 and 522. Since the set of deletions are quite

TABLE 5.5: The number of clients, facilities and edges for each Chromosome (1 to M) in the NA12878 genome.

	Clients	Facilities	Edges		Clients	Facilities	Edges
1	24072	10529	95478	14	10024	4515	59345
2	26500	12210	105243	15	8183	3698	32472
3	20102	9488	83990	16	10811	5031	72084
4	22129	10420	104992	17	10412	4442	43527
5	20426	9164	96047	18	10072	4608	65437
6	18594	8619	91562	19	10395	4099	45610
7	21348	9488	97963	20	8545	3447	47990
8	15602	7248	55815	21	7256	3136	80558
9	12000	5411	49745	22	4635	1905	15652
10	15710	6832	53240	X	13021	6331	40353
11	16956	7667	88284	Y	4097	1020	42534
12	15790	6976	54172	M	200	37	4233
13	10371	4818	35632				

small, the precise recall values have to be considered with care. The overall trend, however, seems to agree with the results on the simulated data set.

Again, with a rather modest loss in recall we are able to increase the precision with respect to truth sets *1000G* and *long-read* from 8.8% to around 69%, and from 9.9% to around 79%, respectively. Similar to the results on the simulated data set, the approximate or optimal solutions our algorithms find for the maximum facility location formulation are similar in terms of recall and precision to the ones returned by the greedy heuristics. In contrast to the greedy approach, however, our algorithms compute these solutions in the order of minutes rather than days.

Similar to the previous analysis, we can also look at the gap in value between our algorithms and the optimal solution returned by the integer program (IP). Inspecting Table 5.6, we can observe that all algorithms produce a feasible solution whose value is close to the optimum. Interestingly, the dependent rounding algorithm doesn't always perform better than the other algorithms. However, both the independent and dependent rounding algorithm often perform better in terms of value than both of the existing greedy algorithms. Once again, the sparse interval graph and lack of conflicts between intervals leads to an instance that in general can be easily solved.

TABLE 5.6: The value of each algorithm, for each chromosome in the NA12878 genome. The relative gap between the value of the optimal solution (IP) and each algorithm is given in brackets (as a percentage).

	IP	Independent		Dependent		Greedy		wGreedy	
1	20918	20905	(0.06)	20906	(0.06)	20838	(0.38)	20835	(0.40)
2	22789	22731	(0.25)	22743	(0.20)	22628	(0.70)	22638	(0.66)
3	17907	17840	(0.37)	17890	(0.10)	17848	(0.33)	17843	(0.36)
4	18618	18616	(0.01)	18616	(0.01)	18496	(0.65)	18512	(0.57)
5	18167	18131	(0.20)	18154	(0.07)	18081	(0.47)	18084	(0.46)
6	16132	16088	(0.27)	16112	(0.12)	16030	(0.63)	16041	(0.56)
7	17886	17798	(0.49)	17856	(0.17)	17770	(0.65)	17794	(0.51)
8	13730	13720	(0.07)	13730	(0.00)	13668	(0.45)	13683	(0.34)
9	10228	10228	(0.00)	10228	(0.00)	10177	(0.50)	10177	(0.50)
10	12737	12737	(0.00)	12737	(0.00)	12617	(0.94)	12625	(0.88)
11	14591	14419	(1.18)	14517	(0.51)	14449	(0.97)	14454	(0.94)
12	13795	13795	(0.00)	13792	(0.02)	13722	(0.53)	13726	(0.50)
13	9156	9149	(0.08)	9077	(0.86)	9126	(0.33)	9130	(0.28)
14	8902	8863	(0.43)	8869	(0.37)	8843	(0.66)	8849	(0.59)
15	7176	7172	(0.06)	7176	(0.00)	7128	(0.68)	7136	(0.56)
16	8978	8978	(0.00)	8976	(0.02)	8857	(1.35)	8877	(1.12)
17	8941	8904	(0.41)	8922	(0.21)	8881	(0.66)	8887	(0.60)
18	8629	8604	(0.29)	8629	(0.00)	8567	(0.71)	8554	(0.86)
19	8536	8536	(0.00)	8536	(0.00)	8426	(1.28)	8432	(1.22)
20	7534	7510	(0.32)	7527	(0.09)	7495	(0.52)	7500	(0.45)
21	5918	5878	(0.68)	5875	(0.73)	5825	(1.58)	5822	(1.63)
22	3967	3950	(0.43)	3960	(0.17)	3940	(0.69)	3950	(0.45)
X	11749	11738	(0.09)	11740	(0.08)	11700	(0.41)	11704	(0.38)
Y	3234	3226	(0.25)	3234	(0.00)	3197	(1.13)	3198	(1.11)
M	158	158	(0.00)	158	(0.00)	158	(0.00)	158	(0.00)

CHAPTER 6

Discussion

We close this work by providing a final discussion of the results thus far. Firstly, we will recap the two main algorithms explored in this thesis (Section 6.1). Afterwards, we discuss the main results of the experiments (Section 6.2). We close the chapter by providing an outline of the current open problems and possible future work (Section 6.3).

6.1 Algorithms

The two algorithms we developed were the SELECT-AND-FILTER and DEPENDENT-ROUNDING algorithms. The first algorithm we presented was the SELECT-AND-FILTER algorithm with a conflict resolution scheme proposed by Feldman (2013). We improved on this scheme by ensuring that we always open a superset of the facilities opened by Feldman's scheme (see Section 4.1.1). The algorithm led to a 0.19-approximation. This algorithm in particular is convenient to implement since it involves solving a linear program, and then running a simple conflict resolution scheme.

The second algorithm, called DEPENDENT-ROUNDING, is based on the technique popularised by Gandhi et al. (2006). It involves gradually shifting the fractional solution returned by (LP) to an integral solution, while still remaining feasible.[1] We adapted this technique to develop an algorithm for the maximum facility location problem. The algorithm yielded some nice properties. For example, the probability that a facility $v \in F$ is open in the final solution is exactly y_v – the value of the facility-variable assigned by (LP). While the algorithm results in a non-constant approximation, it's execution time can be much quicker than the SELECT-AND-FILTER algorithm, as it largely depends on the number of tight points.

[1] This implies that the integral solution is also a feasible solution to (IP).

6.2 Experiments

In this thesis we looked at one application of the maximum facility location problem. Given a donor genome, we aim to map these reads from the donor genome to deletions in the reference genome. From this problem, we can construct it's abstract representation; where facilities are deletions in the reference genome and clients are the reads in the donor genome. These reads could map to multiple places in the reference genome (each having various characteristics), leading to an ambiguous mapping. The crucial observation was that these deletions could be represented as intervals on the real line, completing the reduction to the maximum facility location problem.

The algorithms were tested on two data sets: the Craig Venter genome and the NA12878 genome. In both cases our algorithms performed as well as existing methods, such as the greedy algorithms (Hormozdiari et al., 2010). More specifically, our methods were up to two orders of magnitude faster without sacrificing solution quality (in terms of precision and recall). All of the algorithms also produced solutions that were close to the optimal solution, which is surprising. However further analysis showed that these instances, whilst large, were quite "easy" to solve. This is because the interval graph was very sparse, with few conflicts in between. Even on some of the larger instances (such a chromosome 1 and 2), the DEPENDENT-ROUNDING algorithm only reported there being less than 100 tight points.

6.3 Open Problems and Future Work

There are three main areas for possible future work in this area. The first and foremost priority is developing an algorithm with a better approximation guarantee. In this thesis we developed an algorithm with a 0.19-approximation. On the other hand, we also showed that this problem cannot be approximated in polynomial time within a factor of $(1 - 1/e + \varepsilon)$ for $\varepsilon > 0$ unless P = NP. The value of $1 - 1/e$ is approximately 0.64. Feldman (2013) shows that there is a 0.25-approximation algorithm. Therefore there is a large 'gap' between the existing algorithms and the inapproximability result. However, this doesn't mean that an algorithm with an approximation between 0.25 and 0.64 exists. This is because there is no way to know if our inapproximability result is tight. This was demonstrated in Section 4.3, where we showed that no ρ-approximation polynomial time algorithm exists for $\rho > 1 - 1/e$ unless P = NP. Therefore

the question that remains is whether there exists an algorithm with an optimal approximation ratio of $1 - 1/e$ or a stronger inapproximability result.

The reader may have observed that in Section 4.2 we prove that the DEPENDENT-ROUNDING algorithms leads to a non-constant approximation, but do not state the exact approximation ratio. This is a much harder task. Primarily, it is because the random decisions we make at each step are not independent, and thus we cannot use any of the results from the first algorithm. Therefore a possible area of future work is to analyse the approximation ratio of this algorithm.

In Feldman's work, he develops a contention resolution scheme for the problem of maximising a monotone submodular function subject to a interval graph independence constraint (Feldman, 2013). We then improved on this contention resolution scheme. However the question of whether there is a more efficient contention resolution scheme arises. Rather than just using a simple heuristic to select intervals, perhaps one could incorporate additional data (such as how much a facility contributes to the final solution) in order to make more educated decisions. On the other hand, we could argue that this data was given to us indirectly, as the first step of our algorithm randomly selects a facility v with probability αy_v. This means that if a facility has a large y-value, the linear program is indicating that it is important this facility is kept open in the integral solution.

CHAPTER 7

Conclusion

In this work we have presented two new algorithms for the maximum facility location problem. Both of these algorithms receive a fractional solution from the linear program (Chapter 3) as input and produce an integral solution. These algorithms were discussed in Chapter 4. The first algorithm, which we called SELECT-AND-FILTER, involved randomly selecting facilities from the fractional solution (LP) and removing conflicts. Originally, we used a contention resolution scheme designed by Feldman (2013) to remove conflicts between the intervals. However we then improved on Feldman's scheme, arguing that our new scheme always selects a superset of the facilities that Feldman's method would have selected. This algorithm is attractive since it has a theoretical guarantee (i.e. it will not produce a solution worse than 0.19 times the optimal) and is easy to implement. The easy implementation is primarily due to the fact that one has to only solve a linear program (for which there are many existing libraries) and then implement a simple contention resolution scheme.

The second algorithm, DEPENDENT-ROUNDING, was based on an existing technique known as pipage rounding (see Section 2.3.4) in which one gradually (and randomly) shifts a fractional solution to an integral solution. This algorithm, while slightly less trivial to implement, in some cases can be faster than the SELECT-AND-FILTER algorithm. Since the DEPENDENT-ROUNDING algorithm works with the tight points of the fractional solution, this means that the running time is partially based upon the number of tight points. For instances that have sparse interval graphs and few conflicts, the DEPENDENT-ROUNDING rounding outperforms the SELECT-AND-FILTER algorithm with regards to run time. This was evident in the experiments conducted, as seen in Table 5.1 and Table 5.4 of Chapter 5. We concluded the discussion of the DEPENDENT-ROUNDING algorithm by noting that it admitted a non-constant approximation ratio.

We proved an inapproximability result for the maximum facility location problem in Section 4.3. By reducing the maximum coverage problem to the maximum facility location problem, we showed that there is in fact no polynomial time ρ-approximation algorithm for $\rho > 1 - 1/e$ unless P = NP.

In Chapter 5, we demonstrated how the abstract problem can be represented as a biological problem, where our goal was to map differences between a donor and reference genome. The results demonstrated that our algorithms are fast and effective at producing solutions as good as existing methods. We believe the model is particularly useful for this application since we formulate conflicts between deletions as overlaps of intervals, and do not impose any other assumptions. Thus, the work developed in this thesis can be useful both from an abstract and biological point of view. In particular, we believe this biological formulation will be useful in a wide range of reference-based NGS (next-generation sequencing) data analysis problems. For example, the model we developed could be useful in identifying tumour-specific deletions from matched tumour/normal samples based on their conflict and ambiguous mappings (Wittler and Chauve, 2011) as well as other metagenomic issues.

Bibliography

Alexander A Ageev and Maxim I Sviridenko. 2004. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328.

Ashwinkumar Badanidiyuru and Jan Vondrák. 2014. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1497–1514. SIAM.

Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766.

Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. 2010. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*, volume 10, pages 575–584.

Ken Chen, John W Wallis, Michael D McLellan, David E Larson, Joelle M Kalicki, Craig S Pohl, Sean D McGrath, Michael C Wendl, Qunyuan Zhang, Devin P Locke, et al. 2009. Breakdancer: an algorithm for high-resolution mapping of genomic structural variation. *Nature methods*, 6(9):677–681.

Vasek Chvatal. 1979. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235.

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

Moran Feldman. 2013. *Maximization Problems with Submodular Objective Functions*. Ph.D. thesis, Israel Institute of Technology.

Marshall L Fisher, George L Nemhauser, and Laurence A Wolsey. 1978. An analysis of approximations for maximizing submodular set functions–ii. *Polyhedral combinatorics*, pages 73–87.

Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. 2006. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)*, 53(3):324–360.

Shayan Oveis Gharan and Jan Vondrák. 2011. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1098–1116. SIAM.

Gurobi. 2015. Gurobi.

Fereydoun Hormozdiari, Can Alkan, Evan E Eichler, and S Cenk Sahinalp. 2009. Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes. *Genome*

research, 19(7):1270–1278.

Fereydoun Hormozdiari, Iman Hajirasouliha, Phuong Dao, Faraz Hach, Deniz Yorukoglu, Can Alkan, Evan E Eichler, and S Cenk Sahinalp. 2010. Next-generation variationhunter: combinatorial algorithms for transposon insertion discovery. *Bioinformatics*, 26(12):i350–i357.

David S Johnson. 1973. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49. ACM.

Samir Khuller, Anna Moss, and Joseph Seffi Naor. 1999. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45.

Ariel Kulik, Hadas Shachnai, and Tami Tamir. 2013. Approximations for monotone and non-monotone submodular maximization with knapsack constraints. *Mathematics of Operations Research*, 38(4):729–739.

Ailsa H Land and Alison G Doig. 1960. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520.

Ryan M Layer, Colby Chiang, Aaron R Quinlan, and Ira M Hall. 2014. Lumpy: a probabilistic framework for structural variant discovery. *Genome biology*, 15(6):R84.

Seunghak Lee, Fereydoun Hormozdiari, Can Alkan, and Michael Brudno. 2009. Modil: detecting small indels from clone-end sequencing with mixtures of distributions. *Nature methods*, 6(7):473–474.

Shi Li. 2013. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58.

László Lovász. 1975. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390.

Tobias Marschall, Ivan G Costa, Stefan Canzar, Markus Bauer, Gunnar W Klau, Alexander Schliep, and Alexander Schönhuth. 2012. Clever: clique-enumerating variant finder. *Bioinformatics*, 28(22):2875–2882.

Ryan E Mills, Klaudia Walter, Chip Stewart, Robert E Handsaker, Ken Chen, Can Alkan, Alexej Abyzov, Seungtai Chris Yoon, Kai Ye, R Keira Cheetham, et al. 2011. Mapping copy number variation by population-scale genome sequencing. *Nature*, 470(7332):59–65.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions–i. *Mathematical Programming*, 14(1):265–294.

Aaron R Quinlan, Royden A Clark, Svetlana Sokolova, Mitchell L Leibowitz, Yujun Zhang, Matthew E Hurles, Joshua C Mell, and Ira M Hall. 2010. Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome. *Genome research*, 20(5):623–635.

David B Shmoys, Éva Tardos, and Karen Aardal. 1997. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274. ACM.

Suzanne Sindi, Elena Helman, Ali Bashir, and Benjamin J Raphael. 2009. A geometric approach for classification and comparison of structural variants. *Bioinformatics*, 25(12):i222–i230.

Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. 2011. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 783–792. ACM.

Kai Wei, Rishabh Iyer, and Jeff Bilmes. 2014. Fast multi-stage submodular maximization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1494–1502.

Roland Wittler and Cedric Chauve. 2011. Consistency-based detection of potential tumor-specific deletions in matched normal/tumor genomes. *BMC bioinformatics*, 12(Suppl 9):S21.

Mihalis Yannakakis. 1994. On the approximation of maximum satisfiability. *Journal of Algorithms*, 17(3):475–502.

Kai Ye, Marcel H Schulz, Quan Long, Rolf Apweiler, and Zemin Ning. 2009. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, 25(21):2865–2871.

Jin Zhang, Jiayin Wang, and Yufeng Wu. 2012. An improved approach for accurate and efficient calling of structural variations with low-coverage sequence data. *BMC bioinformatics*, 13(Suppl 6):S6.