Pontifícia Universidade Católica
do Rio de Janeiro

Leonardo Lobo da Cunha da Fontoura

On the Min Distance Superset Problem

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–Graduação em Informática of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor : Prof. Thibaut Vidal
Co–advisor: Prof. Marcus V.S. Poggi de Aragão

Rio de Janeiro
August 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Leonardo Lobo da Cunha da Fontoura

On the Min Distance Superset Problem

Dissertation presented to the Programa de Pós–Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC–Rio, as partial fulfillment of the requirements for the degree of Mestre.

Prof. Thibaut Vidal
Advisor
Departamento de Informática — PUC–Rio

Prof. Marcus V.S. Poggi de Aragão
Co–advisor
Departamento de Informática — PUC–Rio

Prof. Carlile Campos Lavor
Departamento de Matemática Aplicada — UNICAMP

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática — PUC-Rio

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico — PUC–Rio

Rio de Janeiro, August 19th, 2015

Leonardo Lobo da Cunha da Fontoura

Leonardo Lobo da Cunha da Fontoura obtained a bachelor's degree in Mathematics from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio, Rio de Janeiro, Brazil). He earned a scholarship from Capes to do his masters at PUC-Rio.

Bibliographic data

Fontoura, Leonardo Lobo da Cunha da

On the Min Distance Superset Problem / Leonardo Lobo da Cunha da Fontoura; advisor: Thibaut Vidal; co–advisor: Marcus V.S. Poggi de Aragão. — 2015.

48 f. : il. (color.); 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Departamento de Informática, 2015.

Inclui bibliografia.

1. Informática – Teses. 2. Partial Digest Problem. 3. Turnpike Problem. 4. Mapeamento de Sítios de Restrição. 5. Modelagem Matemática. 6. Otimização Quadrática. 7. Otimização Inteira. I. Vidal, Thibaut. II. V.S. Poggi de Aragão, Marcus. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Acknowledgement

I would like to thank my parents, Letícia and Antonio, for their unconditional love and support and my girlfriend, Gabriela, for all the amazing moments we've gone through together.

I would also like to thank CAPES, for supporting me through the entire duration of this degree via a scholarship.

Abstract

Fontoura, Leonardo Lobo da Cunha da; Vidal, Thibaut (Advisor); V.S. Poggi de Aragão, Marcus (Co-Advisor). **On the Min Distance Superset Problem**. Rio de Janeiro, 2015. 48p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The Partial Digest Problem, also known as the Turnpike Problem, consists of building a set of points on the real line given their unlabeled pairwise distances. A variant of this problem, named Min Distance Superset Problem, deals with incomplete input in which distances may be missing. The goal is to find a minimal set of points on the real line such that the multiset of their pairwise distances is a superset of the input.
The main contributions of this work are two different mathematical programming formulations for the Min Distance Superset Problem: a quadratic programming formulation and an integer programming formulation. We show how to apply direct computation methods for variable bounds on top of a Lagrangian relaxation of the quadratic formulation. We also introduce two approaches to solve the integer programming formulation, both based on binary searches on the cardinality of an optimal solution. One is based on a subset of decision variables, in an attempt to deal with a simpler feasibility problem, and the other is based on distributing available distances between possible points.

Keywords

Partial Digest Problem; Turnpike Problem; Restriction Site Mapping; Mathematical Modeling ; Quadratic Optimization; Integer Optimization.

Resumo

Fontoura, Leonardo Lobo da Cunha da; Vidal, Thibaut; V.S. Poggi de Aragão, Marcus. **Sobre o Problema de Superset Mínimo de Distâncias**. Rio de Janeiro, 2015. 48p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O *Partial Digest Problem* (problema de digestão parcial), também conhecido como o *Turnpike Problem*, consiste na construção de um conjunto de pontos na reta real dadas as distâncias não designadas entre todos os pares de pontos. Uma variante deste problema, chamada *Min Distance Superset Problem* (problema de superset de distância mínimo), lida com entradas incompletas em que algumas distâncias podem estar faltando. O objetivo deste problema é encontrar um conjunto mínimo de pontos na reta real, tal que as distâncias entre cada par de pontos contenham todas as distâncias de entrada.

As principais contribuições deste trabalho são duas formulações de programação matemática diferentes para o *Min Distance Superset Problem*: uma formulação de programação quadrática e uma formulação de programação inteira. Mostramos como aplicar um método de cálculo direto de limites de valores de variáveis através de uma relaxação Lagrangeana da formulação quadrática. Também introduzimos duas abordagens diferentes para resolver a formulação inteira, ambas baseadas em buscas binárias na cardinalidade de uma solução ótima. A primeira baseia-se num subconjunto de variáveis de decisão, na tentativa de lidar com um problema de viabilidade mais simples, e o segundo é baseado na distribuição de distâncias entre possíveis pontos disponíveis.

Palavras–chave

Partial Digest Problem; Turnpike Problem; Mapeamento de Sítios de Restrição; Modelagem Matemática; Otimização Quadrática; Otimização Inteira.

Contents

1
Introduction

Computing all pairwise distances from a set of n points is simple. The reverse problem is far less trivial: reconstructing all sets of n numbers on a line given its unordered set of $\binom{n}{2}$ distances. This problem is more commonly known as the *partial digest problem* (PDP). Alternatively, it is also known as the *turnpike problem* (Dakic, 2000), where one is given the pairwise distances of all cities along a highway and has to find their ordering along the road. Despite considerable research efforts, it is still unknown if this problem has a polynomial time algorithm or if it is NP-Complete (Rosenblatt and Seymour, 1982; Lemke and Werman, 1988; Skiena et al., 1990; Zhang, 1994; Dakic, 2000; Daurat et al., 2002, 2005; Nadimi et al., 2011).

The partial digest problem seems to have first appeared relating to phase retrieval problems in X-ray crystallography. Later on, it found an important application in DNA sequencing. We choose to refer to it as the partial digest problem, rather than the turnpike problem, because we will focus on a variant that arises directly from its biological application.

A DNA molecule can be viewed as a string on an alphabet of nucleotides {A, C, G, T}. A *restriction enzyme* is a chemical that cuts DNA at specific sequence patterns of nucleotides, called *restriction sites*. A *digestion experiment* is performed by allowing a restriction enzyme to digest several clones of DNA molecule and then measuring the lengths of the resulting fragments, utilizing a process called *gel electrophoresis*.

If a single restriction enzyme is allowed to completely digest a DNA molecule, then the only information obtained are the distances between any two consecutive restriction sites. Thus, any permutation of the restriction sites is valid in regards to these fragments, as we cannot infer in which order the sites lie. This is called a complete digestion.

If the same enzyme is only allowed to act for a very limited duration, then all fragments between any two restriction sites are obtained. This is where the partial digest problem arises, as now we need to obtain a set of points on a line that generates these pairwise distances exactly, allowing us to find the appropriate positions of known fragments, even without sequencing the whole molecule.

Evidently, experiments and measurements are never perfect. In the case of partial digest experiments, there exists four types of errors (Cieliebak et al.,

2003):

1. Additional fragments: an enzyme might cut at a site that is similar, but not identical to its restriction site. Contamination with unrelated biological material may also happen. This results in additional distances being added to the data.

2. Missing fragments: sometimes a particular restriction site will be left uncut, then only one larger fragment occurs instead of two or more fragments. Also, fragments can remain undetected if they are too small or if their amount is insufficient.

3. Measurement errors: it is next to impossible to determine the exact length of fragments using gel electrophoresis. While accuracies approaching 0.1% are feasible (Skiena et al., 1990), typical error ranges from 2% to 7% of the fragment length (Cieliebak et al., 2003).

4. Multiplicity detection: determining the multiplicity of a distance from its spot in the gel is a non-trivial problem.

These issues tend to complicate the task of retrieving viable reconstructions of the molecule's restriction sites and possible solutions have been addressed in the literature. Skiena, Lemke and Smith (Skiena et al., 1990) proves that the partial digestion problem is strongly NP-complete if additive error bounds are assigned to each distance individually. Skiena and Sundaram (Skiena and Sundaram, 1994) proposed a backtracking algorithm that can deal with measurement errors and few missing fragments. This algorithm has expected running time polynomial in the number of distances, but exponential worst case running time (Zhang, 1994). Several PDP variants were proven to be NP-Hard (Cieliebak et al., 2003).

The second type of error is of particular interest to us. One simple definition of this problem, given in (Cieliebak et al., 2003), is to find the smallest set of points whose pairwise distances contains a given multiset. Intuitively, this means that the objective is to reconstruct a valid set of points that uses the least number of unknown distances. This problem was shown to be NP-hard in (Cieliebak et al., 2003) via a reduction to a problem known as Equal Subsets Sum (Woeginger and Yu, 1992). This problem will be the main focus of this thesis and we will refer to it as the *Min Distance Superset Problem* (MDSP).

A major challenge when dealing with this problem are the high number of symmetrical feasible solutions. Not only there usually exists several solutions

that are equivalent via translation and mirroring, but it is almost always possible to find similar solutions with the same cardinality either via permutation of equivalent distances or changing the order in which points appear.

The contributions of this work are two different mathematical programming formulations, a quadratic programming (QP) formulation and an integer programming (IP) formulation. We show how to apply some of the results found in (Fleischman, 2010) on top of a Lagrangian relaxation of the QP formulation. We also introduce two approaches to solve the IP formulation, both based on binary searches on the cardinality of an optimal solution. One is based on a subset of decision variables, in an attempt do deal with simpler feasibility problem, and the other is based on distributing distances between possible points.

This work is organized in the following way. In chapter 2, we define both the PDP and MDSP, showing some results for both problems, along with a brief review of related works. In Chapter 3, a pseudo-polynomial quadratic formulation for the MDSP is given with its linearization. We also show how to approximate it via Lagrangian relaxation and how to extract additional information about its variables. In Chapter 4, a polynomial integer formulation is built and two ways to solve it are presented. In Chapter 5, we provide experimental results and analysis of the proposed methods and a reasoning behind chosen instances. Finally, a brief conclusion is made in Chapter 6.

2
Partial Digest and Min Distance Superset

2.1
Partial Digest Problem

The problem of reconstructing points in a line, given the distances between any two of them, is known as the *Partial Digestion Problem*. This problem is closely related to the min distance superset problem, as the PDP is a particular case of the MDSP, where no distances are missing. This chapter will consist of a review of related works to both the PDP and MDSP, along with a few results about the former.

Since several pairs of points may have the same distance between them, the input is represented as a *multiset*. A multiset is a set that allows for the repetition of elements. Subtracting an element from a multiset will remove it only once, for example: $\{1,1,1,3,9,12\} - \{1,1,3,4,5\} = \{1,9,12\}$. Curly brackets will denote multisets unless noted otherwise. For the sake of brevity, the multiset $\{q-p \,|\, p,q \in P, p<q\}$ of pairwise distances of a set P will be denoted as ΔP. With this in mind, the partial digest problem can be defined as follows:

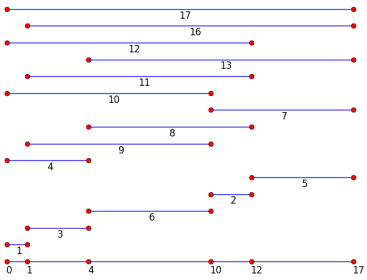
Definition 1 (PDP) *Given a multiset $D = \{d_1,\ldots,d_k\}$ of $k = \binom{n}{2}$ positive integers, is there a set $P = \{p_1,\ldots,p_n\}$ of n points on a line such that $\Delta P \equiv D$?*

To illustrate the problem, let $D = \{1,2,3,4,5,6,7,8,9,10,11,12,13,16,17\}$ be a distance set, known as Bloom's distance set (Skiena et al., 1990). Two possible reconstructions are possible, $P_1 = \{0,1,4,10,12,17\}$ and $P_2 = \{0,1,8,11,13,17\}$, as can be seen in Figure 2.1, that is, $\Delta P_1 \equiv \Delta P_2 \equiv D$.

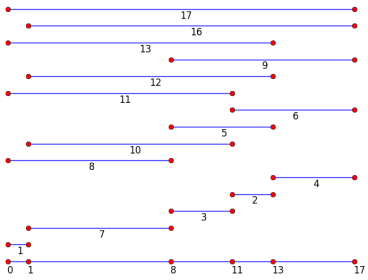
It is important to notice that the two solutions in Figure 2.1 are not *congruent*, as there is no set of rigid motions on the real line that can turn one solution into the other. This means that there is no $a \in \mathbb{R}$ such that $P_1 = P_2 + a = \{p + a \,|\, p \in P_2\}$ or $P_1 = -P_2 + a$ (defined analogously). As congruent sets essentially represent the same reconstruction in the PDP, there is a great interest in *homometric sets*. Two non-congruent sets X, Y are said to be *homometric* if they generate the same distance multisets, $\Delta X \equiv \Delta Y$.

Rosenblatt and Seymour (Rosenblatt and Seymour, 1982) give necessary and sufficient conditions for two sets to have the same distance multiset via

2.0(a): P_1 and ΔP_1.



2.0(b): P_2 and ΔP_2.



polynomial factorization. Given $D = \{d_1, \ldots, d_k\}$, where $k = \binom{n}{2}$, the distance generating function of D is

$$Q(x) = n + \sum_{d \in D} (x^d + x^{-d}) \tag{2-1}$$

By factoring (2-1) over the ring of polynomials with integer coefficients, it is possible to generate all homometric sets that have D as a distance multiset. Later on, Lemke and Werman (Lemke and Werman, 1988) show that this factorization can be done in pseudo-polynomial time depending on the largest exponent of $Q(X)$, which means that the PDP can be solved in pseudo-polynomial time depending on $\max_{d \in D} d$, the largest distance in D.

The maximum possible number of mutually homometric sets of n points, denoted as $H(n)$ is bounded by Skiena, Lemke and Smith in (Skiena et al., 1990):

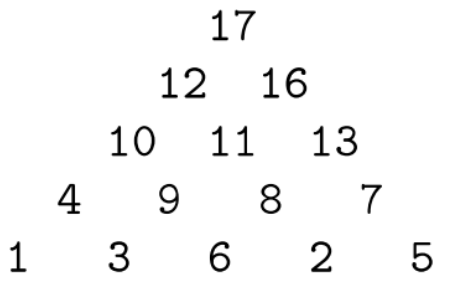
$$H(n) \leq \frac{1}{2} n^{n.12334827}$$

which is valid for all values of n, and $H(n)$ is a power of two. They claim that this suggests that a strongly-polynomial algorithm for the PDP exists, as no NP-complete problem is known with the property that the number of solutions is $2^{o(n^{o(1)})}$.

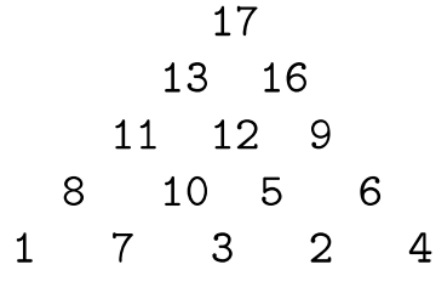
In the same publication, Skiena *et al.* present a combinatorial aspect of the partial digest problem by introducing a backtracking algorithm. This algorithms relies on the fact that the distance multiset D of n points, $P = \{p_1 = 0 < p_2 < \cdots < p_n\}$, can be represented in a triangular matrix or *pyramid*. Let d_{ij} be the distance between the ith and jth points on a line. At the bottom of this pyramid (Figure 2.1) are the distances between consecutive points and the set of distances d_{ij} such that $j - i = l$ are in the lth row of the pyramid. Clearly, d_{1n} is the largest distance in D, as it is the distance between the two farthest points in any solution.

The algorithm works by repeatedly positioning the currently largest

2.0(c): Pyramid for P_1.



2.0(d): Pyramid for P_2.



Algorithm 1 Recursive pseudo-code for Skiena *et al.* backtracking algorithm. Here, D is a multiset of distances, P is a set of points and $dist(p, P)$ returns all distances from p to points in P.

```
1: function PDP_INIT(D)
2:     P = {0, max(D)}
3:     pdp_backtrack(D − {max(D)}, P)
4:
5: function PDP_BACKTRACK(D, P)
6:     if D = ∅ then
7:         output P
8:         return
9:     p = max(D)
10:    if dist(p, P) ⊆ D then
11:        pdp_backtrack(D − dist(p, P), P ∪ {p})
12:    p = max(P) − max(D)
13:    if dist(p, P) ⊆ D then
14:        pdp_backtrack(D − dist(p, P), P ∪ {p})
```

remaining distance of D (Algorithm 1). Notice that this largest distance d must always define a new point p, as it can only be either be the distance d_{1l} or d_{rn} in the pyramid, for some $1 < l, r < n$. Suppose that d is not one of those, but d_{uv} for some $1 < u < v < n$. Then, one of the unpositioned distances d_{un} or d_{1v} is larger than d, which is a contradiction. After choosing this new point p, where $p = 0 + d$ or $p = d_{1n} - d$, it is necessary to check if all distances between p and points in P are available to be used. If the chosen position isn't valid, the algorithm tries the other option for p. If both positions aren't valid, we backtrack one level up and try the other position for that level. This is repeated until either a solution (or all solutions) are found or if the search is exhausted.

In the worst case, Algorithm 1 has a complexity of $\mathbf{O}(2^n n \log n)$, where n is the number of points in the solution instead of the number of distances in the input, but the algorithm works well in practice. Zhang (Zhang, 1994) created a class of PDP instances in which this algorithm takes exponential

time to find a single solution.

This procedure is equivalent to filling the left and right diagonals of the pyramid and then attempting to fill any other position in the pyramid that can be deduced, as both these diagonals directly define points in a solution, since the positions of the first and last point are known beforehand.

In Dakic's thesis (Dakic, 2000), a quadratic programming formulation for the PDP is introduced. Let $D' = \{d_1 < d_1 < \cdots < d_M\}$ be the set of all unique distances in D, and mult(d) be the multiplicity of distance d in D. By considering only solutions P in which the least point is 0, it is clear that all other points in P must be distances that are in D', therefore $P \subseteq D' \cup \{0\}$. By assigning a $0-1$ variable x_d for each $d \in D'$, the PDP can be formulated as the following feasibility problem:

$$\begin{aligned} \sum_{0 \le i < j \le M, \text{ s.t. } j-i=d} x_i x_j &= \text{mult}(d) \qquad d \in D' \\ \sum_{d \in D'} x_d &= n \\ x_i &\in \{0,1\} \qquad 0 \le i \le B \end{aligned} \tag{2-2}$$

Unfortunately, testing for the feasibility of a quadratic $0-1$ program such as (2-2) is a NP-complete problem. To deal with this, Dakic relaxes program (2-2) in a series of increasingly stronger *semidefinite programs* (SDP), which can then be solved in polynomial time. Some classes of PDP instances were shown to be solvable in polynomial time by these semidefinite relaxations, such as instances with an unique solution, instances for which Algorithm 1 only backtracks a constant number of times and the instances proposed by Zhang (Zhang, 1994).

A predictive backtracking algorithm was proposed by Nadimi, Fathabadi and Gantjabesh in (Nadimi et al., 2011). This algorithm is based on a new geometrical interpretation of the PDP. Its search tree is empirically much smaller than the search tree of the backtracking algorithm proposed by Skiena *et al.*, being able to solve many large random instances without having to backtrack at all. It seems to perform well for on the instances proposed by Zhang (Zhang, 1994), however there is no proof that this algorithm can solve the PDP efficiently.

Finally, the work of Daurat, Gérard and Dirat (Daurat et al., 2002) deals with a variation of the PDP called the *Chords Problem*. In this problem, instead of being given the pairwise distance of some points belonging to $\mathbb{N}^d$, one is given their pairwise vector differences. Clearly, the Chords Problem coincides with the PDP when the dimension is 1. Two algorithms are proposed: a variation of Skiena's backtracking algorithm and a variation of the polynomial

factoring algorithm. The first is able to achieve better computational results by postponing ambiguous choices until more information is obtained. It remains unknown if this variation of the backtracking algorithm has exponential worst case complexity, as Zhang's results do not apply. The second algorithm exploits polynomial symmetry in order to factor a distance generating function similar to (2-1) over the polynomials with $\{0, 1\}$ coefficients, solving both the Chords Problem and the PDP in polynomial time. The claim that the PDP is solvable in polynomial time was later retracted in (Daurat et al., 2005), as a different encoding to the PDP problem was assumed, which resulted in an incorrect analysis of the algorithm.

2.2 Min Distance Superset Problem

The Min Distance Superset Problem is a variation of the Partial Digest Problem in which an unknown number of distances between the sampled points are not available, but we still wish to know how the points were arranged. The key assumption is that there can only be missing distances, so every integer in the input was originally a distance between two points in the original structure. This means that the distance multiset of the points we reconstruct must be a superset of the original data. Thus, the Min Distance Superset Problem can be defined as:

Definition 2 (MDSP) *Given a multiset $D = \{d_1, d_2, \ldots, d_k\}$ of k positive integers, find the smallest set $P \subset \mathbb{R}$ such that $D \subseteq \Delta P$.*

Notice that it might be impossible to retrieve the original set of points depending on which distances are missing. For example, consider the set of points $P_0 = \{0, 2, 5, 10\}$ and its distance multiset $\Delta P_0 = \{2, 3, 5, 5, 8, 10\}$. If the input for the MDSP is $D = \Delta P_0 - \{3, 5, 5\}$, then there is no way of inferring the existence of the point 5 without stronger assumptions about the original structure of P_0.

The MDSP is closely related to the PDP as an algorithm for the former can also solve the latter. Any instance of the PDP that has a valid reconstruction must also have a valid reconstruction under the MDSP, since it can be seen as an MDSP instance with no distances missing. Given an instance D of the PDP of cardinality k, then there is a reconstruction for D if, and only if, the optimal solution for the MDSP instance D has size $\frac{1}{2} + \sqrt{\frac{1}{4} + 2k}$. Note that $\frac{1}{2} + \sqrt{\frac{1}{4} + 2k}$ is always a lower bound for the MDSP given an instance D of cardinality k.

Unlike the PDP, there can be no tight general result on how many optimal reconstructions can exist for an arbitrary MDSP instance consisting of k distances. For example, if $D = \{1, 2, 4, \ldots, 2^{k-1}\}$ then only solutions with k points are possible and there are $k!$ optimal solutions corresponding to solutions of the form $P = \left\{0, \sum_{d \in S_1} d, \ldots, \sum_{d \in S_k} d\right\}$, where $\emptyset \subset S_1 \subset S_2 \subset \cdots \subset S_k \subseteq D$. Therefore, nontrivial results regarding the number of optimal solutions for the MDSP will require stronger assumptions regarding the input.

A feasible solution P of a MDSP instance D can be interpreted as an undirected weighted graph $G = (V, E)$. Points in a solution are mapped onto vertices in the graph, so that for each point $p \in P$ there is a vertex $v_p \in V$. Then, there must be exactly one edge with length d for each distance $d \in D$ and it must connect two vertices v_p, v_q such that $\|p - q\| = d$. This is similar to the *pyramid* representation of the PDP. This concept will be the basis for the integer programming formulation introduced in Chapter 4.

Notice that these graph representations are not unique, as there can be many viable ways to place the edges. For instance, let $D = \{1, 2, 2, 4, 6, 10\}$ and $P = \{0, 1, 2, 4, 10\}$. It is simple to verify that at least two different representations are possible: $G_1 = (V, E_1)$ where

$$V = \{v_0, v_1, v_2, v_4, v_{10}\}$$

and

$$E_1 = \{(\mathbf{v_0}, \mathbf{v_1}), (v_0, v_2), (v_2, v_4), (v_0, v_4), (v_4, v_{10}), (v_0, v_{10}))\}$$

and $G_2 = (V, E_2)$ with

$$E_2 = \{(\mathbf{v_1}, \mathbf{v_2}), (v_0, v_2), (v_2, v_4), (v_0, v_4), (v_4, v_{10}), (v_0, v_{10}))\}$$

Furthermore, if there exists a graph representation of (P, D) that is disconnected, then we can find a smaller solution by contracting two appropriate vertices, which must be chosen in a way that no other vertices overlaps. This can be done by contracting the vertex with the lowest position in one of the connected component of the graph and the vertex with the greatest position in another connected component. The graph representation of (P, D) may be disconnected if P is not an optimal solution for D.

The MDSP has more appeared recently in the literature than the PDP. It was defined and proven to be NP-hard by Cieliebak *et al.* in (Cieliebak et al., 2003) (a more thorough exposition can be found in (Cieliebak, 2003)). The proof of NP-hardness is done by reducing a problem known as Equal Sums Subsets (ESS) (3) to the MDSP. Note that the ESS doesn't allow for repeated

numbers in its input, as each set in a solution can consist of one of the repeated numbers.

Definition 3 (Equal Sum Subsets) *Given a set S of k positive integers, are there two non-empty disjoint sets $X, Y \subseteq S$ such that $\sum_{x \in X} x = \sum_{y \in Y} y$?*

The intuition behind the reduction is that if the two disjoint sets in the ESS exists, then the graph representation of an optimal solution to the MDSP with the same input must have a cycle. In this cycle, there are two disconnected paths connecting a point of lower value to a point of greater value and the sum of lengths in both paths must be equal, thus the edges in each path make up each part of the ESS solution. Conversely, any instance of length k of the MDSP that has a solution with cardinality less than $k+1$ must contain contain a cycle. Again, the two paths in the cycle are a solution to the ESS.

This NP-hardness result is further strengthened by considering the t-Min Distance Superset Problem (tMDSP), where t is some parameter specified as a fixed function of $\|D\|$:

Definition 4 (tMDSP) *Given a multiset $D = \{d_1, d_2, \ldots, d_k\}$ of k positive integers, is there a set $P \subset \mathbb{R}$ such that $D \subseteq \Delta P$ and $\|P\| \leq t$?*

This variation of the MDSP is NP-hard (Cieliebak et al., 2003) for any parameter $t = f(\|D\|) = \|D\|^{\frac{1}{2}+\epsilon}$, where $0 < \epsilon < \frac{1}{2}$. It can be considered a tight result, as any solution for the MDSP, tMDSP and the PDP must have cardinality $\Omega(\|D\|^{\frac{1}{2}})$.

2.3 Conclusions

This chapter defined both the partial digest problem and its variation with missing distances, the min distance superset problem. As the PDP is a well studied problem, a brief survey of the literature was provided, which includes several algorithms and theoretical results. Despite the importance of studying PDP variations that take experimental errors into account, there is little research done in any of them. For the MDSP, this includes only proof of its NP-hardness and proof of NP-hardness for the tMDSP.

Two major challenges when dealing with the MDSP are the lack of any theoretical result, other than the NP-hardness proof, and the lack of algorithms to deal with it. These kinds of result could help developing more efficient algorithms for the MDSP. In the following chapters, we work towards introducing integer and quadratic programming formulation for the MDSP, along with some techniques that can be applied to each model. These models,

in turn, have presented even greater challenges, as they can become unwieldy as the input size increases.

3
Methods for a Pseudo-Polynomial Formulation

Quadratically constrained quadratic programming (QCQP) play an important role as a modelling tool for many problems. The downside is that, while linear programs can be solved efficiently, QCQP is NP-hard. Particularly, Dakic (Dakic, 2000) has used $0-1$ QCQP to great effect when modelling the partial digest problem. A quadratically constrained quadratic $0-1$ program is an optimization problem defined as:

$$\begin{array}{lrcll} \min\limits_{x} & \frac{1}{2}x^tQ_0x + {\ell_0}^t x & & & \\ \text{s.t.} & \frac{1}{2}x^tQ_ix + {\ell_i}^t x & \geq & b_i & i = 1, \ldots, k \\ & x_j & \in & \{0,1\} & j = 1, \ldots, n \end{array} \tag{3-1}$$

It is assumed that all quadratic forms Q_i in (3-1) are symmetric. To understand this assumption, notice that for any square matrix Q, $x^tQx = x^tQ^tx$. Let $Q' = \frac{Q+Q^t}{2}$. Then, $x^tQ'x = \frac{x^tQx+x^tQx}{2} = x^tQx$, allowing all non-symmetric matrix to be replaced by an equivalent symmetric matrices.

A feature of quadratic $0-1$ program is the ability to directly model logical AND and OR. An AND between binary variables is multiplication and the OR, addition. For example, suppose an arbitrary QCP has a constraint $xy + zw \geq 2$. Such constraint can only ever be satisfied if all variables have a value of 1, and fails when any of them is 0. This can be exploited in conjunction with the notion of representing feasible solutions for the MDSP as graphs.

Recall that it is possible to find an initial feasible solution for any given MDSP instance, simply by "chaining" the input distances: if $D = \{d_1, \ldots, d_k\}$, then $P = \{0, d_1, d_1 + d_2, \ldots, \sum_{d \in D} d\}$ is a feasible solution of cardinality $k+1$. Therefore, it is unnecessary to consider solutions that have a diameter greater than $B = \sum_{d \in D} d$, as they will necessarily have cardinality greater than $k+1$. Then, the MDSP can be formulated as:

$$\begin{array}{lrcll} \min\limits_{x} & \sum_{i=0}^{B} x_i & & & \\ \text{s.t.} & \sum\limits_{0 \leq i < j \leq B, \text{ s.t. } j-i=d} x_ix_j & \geq & \text{mult}(d) & d \in D' \\ & x_i & \in & \{0,1\} & 0 \leq i \leq B \end{array} \tag{3-2}$$

where D' is the set of unique distances in an instance D and mult(d), $d \in D$ is the multiplicity of the value d, that is, how many times it appears in the multiset of integers D. In this formulation, each variable x_p represents a fixed

integer on a line from 0 to B, having a value of 1 if, and only if, p is part of a solution. A multiplication between two points x_p and x_{p+d} models an edge of length d connecting points p and $p + d$. For each distance d in D' there is a constraint that forces the distance multiset of a solution to use the distance d at least as many times it appears in the input. Finally, the objective minimizes the sum of points in the solution. The indices of the variables that have value 1 are a minimal set of points P such that $D \subseteq \Delta P$.

Despite the simplicity of such formulation, there is a downside: the number of variables is polynomial in the sum of the distances in the input. This means that the size of this formulation is pseudo-polynomial, that is, its size is exponential on the size of the binary representation of the input.

Algorithm 2 Determines which points can be part of a solution. Note that C, T and P are sets, with their usual operations.

```
1: function VALID_POINTS(distances)
2:     C = {0}
3:     for d ∈ distances do
4:         T = {}
5:         for p ∈ C do
6:             T = T ∪ {p − d, p + d}
7:         C = C ∪ T
8:     P = {p | p ∈ C, p ≥ 0}
9:     return P
```

The number of variables can be reduced in two ways: dividing all integers in D by their greatest common divisor or using the procedure shown in algorithm 2. Note that the greatest common divisor method will only work if no pair of numbers in D are relatively prime, as their greatest common divisor will be 1. Algorithm 2 first calculates all possible linear combinations of input distances, with the added constraint that the only coefficients allowed are -1, 0 and 1. Then, it removes all combinations that are less than 0 and returns the resulting set of points.

The idea is that no graph representation of an optimal solution can be disconnected, therefore it is unnecessary to consider points that cannot be reached from the 0 using only distances from D (going forward or backward). This method can lead to a substantial decrease in the size of the formulation if the distances are sparse enough, however it remains exponential on the length of D, as up to $\frac{3^n}{2}$ variables can still exist.

For example, let $D = \{5, 8, 13, 22, 29\}$. If the variables are not filtered with algorithm 2, then 78 variables are necessary, against 56 after applying it. On the other hand, if $D = \{1, 1, 4, 15, 27, 40\}$, then the savings are minimal: 89 variables without filtering against 87. As shown by this example, Algorithm

2 works better if the input is sparse, in the sense that smaller numbers cannot be constructed by summation and subtraction of integers in the input.

3.1 Nonconvexity of the quadratic formulation

One thing to note is that available QCQP solvers, commercially or otherwise, usually only solve QCQPs in which all quadratic constraints are convex. This means that all Q_i matrices in (3-1) must be definite positive, which unfortunately isn't always the case for the quadratic program (3-2). In this case, there are two options to solve this problem: change the offending matrices in such way that they become positive definite, or linearize the formulation.

3.1.1 Convexification of offending constraints

It is always possible to force a constraint in a quadratic $0-1$ program to be convex. Observe the fact that if x_i is a variable in such a program, then $x_i^2 = x_i$ and $x_i^2 - x_i = 0$. Let $\frac{1}{2}x^tQx + \ell^t x \geq k$ be a quadratic constraint and $\eta \in \mathbb{R}^n$. Then,

$$\begin{aligned} & \frac{1}{2}x^tQx + \ell^t x \\ = \; & \frac{1}{2}x^tQx + \ell^t x + \sum_{i=1}^{n} \eta_i(x_i^2 - x_i) \\ = \; & \frac{1}{2}x^t(Q + 2\,\mathrm{Diag}(\eta))x + (\ell^t + \eta^t)x \end{aligned}$$

where $\mathrm{Diag}(\eta)$ is the square matrix with η in its diagonal and 0 everywhere else. By increasing each η_i, it is possible to turn a non-convex constraint convex. It is important to remark the this doesn't alter the problem in any way if integrality constraints are maintained. However, the solution of a linear relaxation may vary with η. It is possible to optimize the value of η in order to get the best possible optimal value from a linear relaxation in this case, by solving an appropriate semidefinite program (Fleischman, 2010).

3.1.2 Applying the Reformulation-Linearization Technique

The other option is to apply the reformulation-linearization technique (RLT). This family of techniques are usually used to produce tighter relaxation of nonconvex problems at the expense of a larger number of variables

and constraints. In the case of a quadratic $0-1$ program, it is possible to linearize the problem while maintaining variable integrality constraints, obtaining an integer program. A thorough explanation of RLT along with several applications in discrete and continuous nonconvex optimization can be found in (Sherali and Adams, 2013), while a more concise explanation of the basic technique can be found in (Anstreicher, 2009).

A simple application of RLT on a QCP is based on using products of lower and upper bound linear constraints of the original x_i variables to obtain valid linear inequality constraints on new variables y_{ij}. In the case of (3-2), it is unnecessary to create a new product variable y_{ij} for each pair x_i and x_j, as only certain products will appear in the constraints.

Notice that (3-2) has integrality constraints on each x_i, and contains no linear constraints at all. However, these integrality constraints imply that $x_i \geq 0$ and $x_i \leq 1$ are valid for all x_i variables. Adding such constraints to (3-2) doesn't change the problem. Multiplying each constraint involving x_i and x_j, if both of them appear in a quadratic constraint, and replacing the term $x_i x_j$ by a new variable y_{ij}, the following constraints are obtained:

$$\begin{aligned} y_{ij} &\geq 0 \\ y_{ij} - x_i - x_j &\geq -1 \\ y_{ij} - x_i &\leq 0 \\ y_{ij} - x_j &\leq 0 \end{aligned}$$

If integrality constraints for the x_i are maintained, then y_{ij} can be 1 if, and only if, the product $x_i x_j$ is also 1. However these constraints are also valid in the case of a linear relaxation. Applying this to (3-2), the following IP is obtained:

$$\begin{array}{lrcll} \min\limits_{x} & \sum_{i=0}^{B} x_i & & & \\ \text{s.t.} & \sum\limits_{0\leq i<j\leq B, \text{ s.t. } j-i=d} y_{ij} & \geq & \text{mult}(d) & d \in D' \\ & y_{ij} - x_i - x_j & \geq & -1 & 0 \leq i < j \leq B \\ & y_{ij} - x_i & \leq & 0 & 0 \leq i < j \leq B \\ & y_{ij} - x_j & \leq & 0 & 0 \leq i < j \leq B \\ & y_{ij} & \geq & 0 & 0 \leq i < j \leq B \\ & x_i & \in & \{0,1\} & 0 \leq i \leq B \end{array} \tag{3-3}$$

The advantage of using this new integer programming model is that there is a wide range of commercial software than can be used to solve it, in contrast to the less available quadratic constrained and nonlinear program solvers. On

the other hand, implicit information contained in the original formulation is lost, which could have been used to strength the model. The performance of this model will be investigate in Chapter 5.

3.2 Lagrangian Relaxation

Lagrangian relaxation is a widely used approximation method for hard optimization problems. It applies generally to all types of mathematical programs: convex, nonconvex, linear, nonlinear, continuous or discrete. Approximations are used to bound the possible values of an optimal solution: the tighter the bounds the better. A Lagrangian relaxation to (3-2) will be introduced, seeking to achieve better approximations than just relaxing integrality constraints in (3-2) and (3-3). It will also be useful in obtaining information about individual variables by applying the methods described in Fleischman's work (Fleischman, 2010).

A lagrangian relaxation of a QCQP consists of attaching Lagrange multipliers to a set of constraint and relaxing them into the objective function. The resulting simpler problem will then be solved exactly. Consider the following relaxation of formulation (3-2), where $\lambda \geq 0$ is fixed:

$$\begin{aligned} \min_{x} \quad & \textstyle\sum_{i=0}^{B} x_i + \sum_{d \in D'} \lambda_d \cdot \left(\mathrm{mult}(d) - \sum_{0 \leq i < j \leq B, \text{ s.t. } j-i=d} x_i x_j \right) \\ \text{s.t.} \quad & \sum_{0 \leq i < j \leq B, \text{ s.t. } j-i=d} x_i x_j \geq \mathrm{mult}(d) \qquad d \in D' \\ & x_i \in \{0,1\} \qquad 0 \leq i \leq B \end{aligned} \tag{3-4}$$

The optimal value of formulation (3-4), for any $\lambda \geq 0$, is clearly a lower bound for (3-2), since

$$\mathrm{mult}(d_k) - \sum_{0 \leq i < j \leq B, \text{ s.t. } j-i=d} x_i x_j \leq 0$$

Thus, it is possible to relax (3-4) again into

$$\begin{aligned} \mathcal{L}(\lambda) = \min_{x} \quad & \textstyle\sum_{i=0}^{B} x_i + \sum_{d \in D'} \lambda_d \cdot \left(\mathrm{mult}(d) - \sum_{0 \leq i < j \leq B, \text{ s.t. } j-i=d} x_i x_j \right) \\ \text{s.t.} \quad & x_i \in \{0,1\} \qquad 0 \leq i \leq B \end{aligned} \tag{3-5}$$

obtaining a lagrangian relaxation of (3-2) (for programs with more constraint sets, there can exist an exponential number of different relaxations), which also provides valid lower bounds for any $\lambda \geq 0$, since removing constraints from a minimization problem can only reduce the value of the optimal solution.

Choosing appropriate values for the lagrange multipliers is of key importance in terms of quality of lower bounds. Particularly, we are interested in

finding the λ which gives the maximum possible lower bound. This involves finding multipliers corresponding to:

$$\max_{\lambda \geq 0} \mathcal{L}(\lambda)$$

which is called the lagrangian dual program of (3-2). Ideally, the optimal value of this program is equal to the optimal value of the original problem, however this seems to rarely happen. A typical way of solving this maximization problem heuristically is via *subgradient optimization*. Subgradient optimization is an iterative procedure to systematically generate lagrangean multipliers from an initial set of multipliers. The basic algorithm for the subgradient optimization of (3-5), considering an instance D of length n, is as follows:

1. Let $U = n + 1$ be the current best upper bound for the original problem, in this case, the trivial solution of chaining distances. Choose an initial lagrangian multipliers $\lambda_d \geq 0$, $d \in D'$, one multiplier for each unique distance in D.

2. Solve (3-5) with the current set of λ to get a solution $\bar{x}_i$ of value L.

3. Define the subgradients G_d, $d \in D'$ as the value of each relaxed constraint:
$$G_d = \text{mult}(d) - \sum_{0 \leq i < j \leq B, \text{ s.t. } j-i=d} \bar{x}_i \bar{x}_j$$

4. Define a step size S, with an arbitrarily defined parameter $0 < \pi \leq 2$, as:
$$S = \pi \frac{(U - L)}{\sum_{d \in D'} (G_d)^2}$$

5. Update each λ_d:
$$\lambda_d = \max(0, \lambda_d + S \cdot G_d)$$
and if termination is not triggered, then go to step 2.

It is usual to terminate this procedure after a fixed number of iterations or by reducing the value of π during the iteration above, terminating when π is small enough.

Directly using the Lagrangian relaxation without acquiring better upper bounds than the trivial one has provided low-quality lower bounds. Experiments have shown that lower bounds provided by the subgradient optimization are no better than the ones provided by simply relaxing integrality constraints on formulation (3-3). However, the CPU time for the subgradient optimization

seems to be lower than solving the linear relaxation based on computational experiments.

3.3 Pruning technique

In (Fleischman, 2010), Fleischman investigates what information can be derived from a convex unconstrained quadratic $0-1$ program when an upper bound U is known. It turns out that it is possible to find the value of some variables for any feasible solution with a value better than U, along with lower bounds for the size of the sets containing all variables equal to 0 and 1. Evidently, such method cannot work directly on (3-2), as it contains several constraints. However, this method can also be applied when dealing with the lagrangian relaxation (3-5).

Consider the following quadratic unconstrained $0-1$ problem:

$$\begin{array}{lll} \min\limits_{x} & x^tQx + \ell^t x & \\ \text{s.t.} & x_j \in \{0,1\} & j = 1,\ldots,n \end{array} \tag{3-6}$$

If an upper bound U to (3-6) is known, then the following problem consisting of a linear objective function and a single quadratic constraint can be solved:

$$\begin{array}{lll} Z = \min\limits_{x} & c^t x & \\ \text{s.t.} & \frac{1}{2}x^tQx + \ell^t x \leq U & \end{array} \tag{3-7}$$

Since relaxing the integrality constraints of (3-6) can only yield lower bounds, then Z must also be a lower bound of $c^t\bar{x}$, where $\bar{x}$ is an optimal solution to (3-6). Solving (3-7) with different vectors c can provide many insights into $\bar{x}$, since Z must be the same for any solutions of value less than U, specially optimal solutions. Some of these are:

- If $c = (0,\ldots,0,1,0,\ldots,0) = e_i$, the ith vector in the canonical basis, and $Z > 0$, then $x_i > 0$, which translates to $x_i = 1$ in the optimal solution because of the integrality constraints.
- Analogously, if $c = -e_i$ and $Z > -1$, then $x_i = 0$. To see this, simply substitute $Z = -{e_i}^t x$ in $Z > -1$ to arrive at $x_i < 1$.
- If $c = \mathbb{1}$, the one vector, then Z is a lower bound for the cardinality of the set $\{i \,|\, x_i = 1\}$.
- Finally, if $c = -\mathbb{1}$, then $n + Z$ is a lower bound for the cardinality of the set $\{i \,|\, x_i = 0\}$.

This information can be used to simplify problems by eliminating variables and in branch and bound environments for pruning purposes. The major

appeal of this method is that it can be performed in polynomial time in the case which the matrix Q is positive definite. For the full proof on why this is possible, see chapter 5 of (Fleischman, 2010).

To solve (3-7) for a given vector c, first calculate

$$\mu = \sqrt{\frac{2U + \ell^t Q^{-1} \ell}{c^t Q^{-1} c}} \tag{3-8}$$

and then substitute μ in

$$x^* = Q^{-1}(\ell - \mu c) \tag{3-9}$$

Computing the Cholesky decomposition of a positive definite real matrix can be done in $\mathbf{O}(n^3)$ and is an extremely efficient procedure in practice. After computing the Cholesky decomposition of Q, it is possible to compute (3-9) for each c in $\mathbf{O}(n^2)$. Another interesting feature of this method is that performing these computations for different c's is an embarrassingly parallel problem, allowing for nearly linear speed up on the number of processors. We will call this the *Pruning technique*.

3.4 Applying the Pruning Technique to the Lagrangian Relaxation

We will show how to apply the pruning technique to the lagrangian relaxation (3-5). Since the matrix in the objective function of (3-5) is not always positive definite, it is necessary to modify the matrices in (3-2) before applying the lagrangian relaxation. First, lets rewrite (3-5) in the following way:

$$\begin{aligned} \min_x \quad & \tfrac{1}{2} x^t \left(\sum_{d \in D'} - \lambda_d Q_d \right) x + \sum_{i=i}^{B} x_i + \sum_{d \in D'} \lambda_d \, \text{mult}(d) \\ \text{s.t.} \quad & x_i \in \{0,1\} \quad 0 \leq i \leq B \end{aligned} \tag{3-10}$$

where Q_d is a matrix of dimensions $(B+1) \times (B+1)$ with both its $\pm dth$ sub-diagonals equal to 1 and everything else equal to 0.

It suffices to notice that for each matrix $-Q_d$, the matrix $-Q_d + \eta \cdot I$ will always be positive definite for a large enough $\eta \in \mathbb{R}$. Empirically, it seems that $\eta = 2$ is large enough for this purpose. Therefore, a positive weighted sum of these matrices is also positive definite. By simply changing constraints

$$\sum_{\substack{j-i=d \\ 0 \leq i < j \leq B}} x_i x_j \geq \text{mult}(d), \quad d \in D'$$

of formulation (3-2) into the identical (due to variables being $0-1$) constraints:

$$\sum_{\substack{j-i=d_k \\ 0\leq i<j\leq B}} x_i x_j - \sum_{i=0}^{B}(x_i^2 - x_i) \geq \text{mult}(d), \quad d \in D' \tag{3-11}$$

The resulting lagrangian relaxation of (3-2) after the constraint modification (3-11) is convex:

$$\begin{array}{ll} \min\limits_{x} & \frac{1}{2}x^t \left[\sum\limits_{d\in D'} \lambda_d \left(-Q_d + 2I\right)\right] x + {b_\lambda}^t x + c_\lambda \\ \text{s.t.} & x_i \in \{0,1\} \quad 0 \leq i \leq B \end{array} \tag{3-12}$$

where $b_\lambda = \left(1 - \sum_{d\in D'} \lambda_d\right) \cdot \mathbb{1}$ is a vector and $c_\lambda = \sum_{d\in D'} \lambda_d \,\text{mult}(d)$ is a constant, both depending on the current lagrangian multipliers. Now, it is possible to indirectly apply (3-7) to the original problem via the Lagrangian relaxation (3-12):

$$\begin{array}{lll} Z_\lambda = \min\limits_{x} & c^t x & \\ \text{s.t.} & \frac{1}{2}x^t \left[\sum\limits_{d\in D'} \lambda_d \left(-Q_d + 2I\right)\right] x + {b_\lambda}^t x & \leq \quad U - c_\lambda \end{array} \tag{3-13}$$

Despite this being done on top of a relaxation of a Lagrangian relaxation, the same information described in section 3.3 can be extracted from (3-13), since $Z_\lambda \leq c^t x_\lambda^* \leq c^t \bar{x}$, where x_λ^* is the optimal solution to the Lagrangian relaxation and $\bar{x}$ is the optimal solution to the original problem. Problem (3-13) can be solved either during or after the subgradient optimization, simply by changing the current multipliers.

3.5 Conclusions

This chapter has introduced a quadratically constrained $0-1$ formulation (3-2) for the MDSP, which is pseudo-polynomial in the number of variables. Since its constraints are not convex in general, it was shown how to apply the reformulation-linearization technique to it, yielding an equivalent integer $0-1$ program (3-3). After that, a Lagrangian relaxation (3-5) of (3-2) was introduced and, finally, it was shown how to apply the pruning technique of (Fleischman, 2010) to the original quadratic formulation via the Lagrange representation.

The quadratic model (3-2) is remarkably simple and contains a lot of implicit information due to variable multiplication. However, a pseudo-polynomial number of variables makes this model a large-scale model even for small instances. It remains an open question whether a similar quadratic formulation for the MDSP with a polynomial number of variables can exist or not. A few possible paths leading to this goal would be pruning points that are

part of symmetrical solutions or identifying positions created from distances that are not part of a larger path in the graph representation of any solution, and thus, could have been placed anywhere. In order to deal with this, an integer programming formulation with polynomial size will be presented in the next chapter.

4
Solving an Integer Formulation

The downside of the quadratic model presented in Chapter 3 is that we are not to determine enough points that cannot be part of an optimal solution and the necessity to represent the existence of each of those points as a binary variable. In this chapter, we will show how to avoid having to represent each possible point in a solution, using a integer programming formulation for the MDSP. Two methods for solving this IP formulation using binary search on the number of points in a optimal solution are also introduced.

Let $D = \{d_1, \ldots, d_k\}$ be a MDSP instance. Then, an optimal solution P must have its cardinality n be between $\left\lceil \frac{1}{2} + \sqrt{\frac{1}{4} + 2k} \right\rceil \leq n \leq k + 1$, where the lower bound is true because at least that many points are necessary to generate k pairwise distances and the upper bound is achieved by chaining distances, as explained in Chapter 3. We will now build the IP formulation gradually.

Recall the graph representation of a feasible solution of the MDSP presented in Section 2.2 and let $G = (V, E)$ be such a graph for P. Note that $|V|$ has the same bounds as $|P|$. We introduce integral variables $p_i \geq 0$, $p_i \in \mathbb{Z}$, $0 \leq i \leq k$, the coordinates of each vertex on the real line. As always, the first vertex is fixed to the origin, and thus $p_0 = 0$. Without loss of generality, we can impose an ordering on the p_i variables, along with making sure no vertices can overlap, with the following constraint:

$$p_i + 1 \leq p_{i+1}, \quad 0 \leq i < k \tag{4-1}$$

The next step is to introduce variables that can represent distances. Since any distance in D can be assigned between any two vertices in V, it is necessary to introduce binary variables $x_{d,i,j} \in \{0, 1\}$, for all $d \in D', 0 \leq i < j \leq k + 1$, where D' is the set of all unique distances in D. Each $x_{d,i,j}$ represents if the distance d has been used between vertices i and j. Evidently, only up to one distance may be used between any pair of vertices, leading to the constraint:

$$\sum_{d \in D'} x_{d,i,j} \leq 1, \quad 0 \leq i < j \leq k \tag{4-2}$$

and each unique distance $d \in D'$ must be used exactly $\text{mult}(d)$ times:

$$\sum_{i=0}^{k-1} \sum_{j=i+1}^{k} x_{d,i,j} = \text{mult}(d), \quad d \in D' \tag{4-3}$$

If some variable $x_{d,i,j}$ is equal to 1, then the distance between p_i and p_j has to be equal to d. This can be achieved through the constraints

$$p_j \leq p_i + d + B(1 - x_{d,i,j}), \quad 0 \leq i < j \leq k, d \in D' \tag{4-4}$$

$$p_j \geq p_i + d \cdot x_{d,i,j}, \quad 0 \leq i < j \leq k, d \in D' \tag{4-5}$$

where $B \geq \sum_{d \in D} d$ is a number larger than the span of any solution with $k+1$ points or less. For any i, j, d, Constraint (4-4) forces p_j to be at most $p_i + d$ if $x_{d,i,j}$ is one, while not restraining p_j position in respect to p_i if $x_{d,i,j}$ is zero, as B should be a large number. Likewise, constraint (4-5) forces p_j to be at least $p_i + d$ if $x_{d,i,j}$ is one and doesn't restrain p_j when $x_{d,i,j}$ is zero, since Constraint (4-1) is tighter in this case. These two constraints ensure that the distance between p_i and p_j is equal to d if $x_{d,i,j}$ is one, while not placing any additional constraints when $x_{d,i,j}$ is zero.

It is possible to tighten Constraint (4-4) by changing the constant B into a constant B_{j-i}, depending on how many distances can be placed between p_i and p_j, that is equal to the sum of the largest $j - i$ distances in D.

Note that the maximum number of vertices have been introduced in the model so far, while it is clear that the optimal solution for many instances do not require all these points. To solve this, we introduce a new set of binary variables $z_i \in \{0, 1\}$, $0 \leq i \leq k$. Each variable z_i is set to 1 if, and only if, the vertex p_i is part of the solution. Since p_0 is fixed at the origin and it is always in the solution, z_0 can be fixed as 1. Then, all active vertices are clustered to the lower indices, through the constraint set

$$z_i \geq z_{i+1}, \quad 0 \leq i < k \tag{4-6}$$

If $x_{d,i,j} = 1$ for some d, then p_i and p_j must be part of the solution and $z_i = z_j = 1$. This effect can be achieved by slightly modifying constraint (4-2) into

$$\sum_{d \in D'} x_{d,i,j} \leq z_j, \quad 0 \leq i < j \leq k, \tag{4-7}$$

allowing a distance to be used between p_i and p_j if, and only, if $z_j = 1$. Constraint (4-7) is sufficient to achieve the desired effect, since constraint (4-6) will force $z_i = 1$ as well.

Finally, the objective is to minimize the sum $\sum_{i=0}^{k} z_i$, the number of active vertices. Thus, we arrive at the following formulation for the MDSP:

$$
\begin{aligned}
\min_{x,p,z} \quad & \sum_{i=0}^{k} z_i \\
\text{s.t.} \quad & \sum_{d \in D'} x_{d,i,j} && \leq && z_j && 0 \leq i < j \leq k \\
& \sum_{i=0}^{k-1} \sum_{j=i+1}^{k} x_{d,i,j} && = && \text{mult}(d) && d \in D' \\
& p_j && \geq && p_i + d \cdot x_{d,i,j} && 0 \leq i \leq k, d \in D' \\
& p_j && \leq && p_i + d + B(1 - x_{d,i,j}) && 0 \leq i \leq k, d \in D' \\
& p_i + 1 && \leq && p_j && 0 \leq i < j \leq k \\
& z_i && \geq && z_{i+1} && 0 \leq i < k \\
& x_{d,i,j} && \in && \{0,1\} && 0 \leq i < j \leq k, d \in D' \\
& p_i && \in && \mathbb{Z} && 0 \leq i \leq k \\
& z_i && \in && \{0,1\} && 0 \leq i \leq k
\end{aligned}
\tag{4-8}
$$

This integer programming formulation for the MDSP is clearly more complex than the quadratic formulation (3-2). However, both the number of variables and constraint present in (4-8) are bounded from above by polynomials on the cardinality of the input multiset D, instead of depending on the values which D contains. This means that the formulation is always the same size for different inputs of same size, unlike formulation (3-2), which can achieve very large scales even for small inputs, if they contain large values.

It is worth noting that while this formulation has a polynomial size on the input, its size is still relatively large: model (4-8) contains

$$\frac{1}{2}k(k+1)\|D'\| \leq \frac{1}{2}k(k+1)\|D\|$$

variables of type $x_{d,i,j}$, plus $k+1$ variables for each type p_i and z_i, and it will also contain a total of

$$k(k+1)\left(1 + \|D'\|\right) + k + \|D'\|$$

constraints (not counting variable bounding constraints). In the worst case, if all distances in D are unique, then model (4-8) will have a cubic quantity of both variables and constraints. For example, if $\|D\| = 20$ and all distances are unique, then there will be a total of 4200 variables and 8860 constraints, while if $\|D\| = 100$, there will be a total of 505000 variables and 1020300 constraints.

4.1 Methods for the Integer Formulation

The integer formulation presented is polynomially sized, but it will quickly become a large scale problem as the size of the input increases. In order to deal with this problem, we propose two methods of solving model (4-8), both involving a binary search on the number of points in the optimal solution and solving different variations of model (4-8) that completely removes the decision variables z_i.

The first method is based on trying to find increasingly smaller solutions for the MDSP instance, until no smaller solution can be found. A solution for the MDSP that uses up to $t+1$ points can be described as the following region:

$$\begin{array}{rcll}
\displaystyle\sum_{d\in D'} x_{d,i,j} & \leq & 1 & 0 \leq i < j \leq t \\
\displaystyle\sum_{i=0}^{t-1}\sum_{j=i+1}^{t} x_{d,i,j} & = & \text{mult}(d) & d \in D' \\
p_j & \geq & p_i + d \cdot x_{d,i,j} & 0 \leq i \leq t, d \in D' \\
p_j & \leq & p_i + d + B(1 - x_{d,i,j}) & 0 \leq i \leq t, d \in D' \\
p_i + 1 & \leq & p_j & 0 \leq i < j \leq t \\
x_{d,i,j} & \in & \{0,1\} & 0 \leq i < j \leq t, d \in D' \\
p_i & \in & \mathbb{Z} & 0 \leq i \leq t
\end{array} \tag{4-9}$$

where t is some arbitrary number less or equal to k. This formulation removes the need of the z_i variables, considering all points as part of the solution, even points that use no distances whatsoever. Finding a feasible solution in (4-9) means that it is possible to assign all distances in D as distances between $t+1$ points in $\mathbb{Z}$. If this region is empty, then there is no way to assign those distances as the distances between $t+1$ integer points or less.

These observations lead to Algorithm 3, a binary search on the number of points in the solution. Recall that the lower bound for any instance of size k is $\left\lceil \frac{1}{2} + \sqrt{\frac{1}{4} + 2k} \right\rceil$ and a trivial upper bound of $k+1$. These bounds are the endpoints of the binary search. The binary search iteratively solves (4-9) with t equal to the middle of the current end points. If a feasible solution was found, then the upper bound is set as the current mid, since now it is known that a solution with that many points can exist, but it isn't known if a solution with less points exists yet. If no feasible solution is found, then the lower bound is updated to $mid+1$, as there cannot exist any solution with less than $mid+1$ points. After the loop, it simply outputs the latest solution found. This is sufficient, as the final iteration of the binary search already proved that any solution with one less point cannot exist.

Algorithm 3 Binary search algorithm based on models (4-9) and (4-10).

```
1: function BS-MDSP(D)
2:     k = length(D)
3:     lb = ⌈1/2 + √(1/4 + 2k)⌉
4:     ub = k + 1
5:     while lb < ub do
6:         mid = ⌊(lb + ub)/2⌋
7:         solve (4-9) or (4-10) for D with t = mid
8:         if a feasible MDSP solution was found then
9:             save the solution
10:            ub = mid
11:        else
12:            lb = mid + 1
13:    output the latest solution found
```

Model (4-9) can be further reduced into a formulation that relies on distributing as many distances as possible among a fixed number of points, instead of checking if a region is empty or not. Such a model can be described as follows:

$$
\begin{aligned}
\max_{x,p} \quad & \sum_{d\in D}\sum_{i=0}^{t-1}\sum_{j=i+1}^{t} x_{d,i,j} & & & \\
\text{s.t.} \quad & \sum_{d\in D'} x_{d,i,j} & \leq & 1 & 0 \leq i < j \leq t \\
& \sum_{i=0}^{t-1}\sum_{j=i+1}^{t} x_{d,i,j} & \leq & \text{mult}(d) & d \in D' \\
& p_j & \geq & p_i + d \cdot x_{d,i,j} & 0 \leq i \leq t, d \in D' \\
& p_j & \leq & p_i + d + B(1 - x_{d,i,j}) & 0 \leq i \leq t, d \in D' \\
& p_i + 1 & \leq & p_j & 0 \leq i < j \leq t \\
& x_{d,i,j} & \in & \{0,1\} & 0 \leq i < j \leq t, d \in D' \\
& p_i & \in & \mathbb{Z} & 0 \leq i \leq t
\end{aligned}
\tag{4-10}
$$

In this formulation, the constraints dealing with p_i variables remains the same. The key difference is that constraint (4-3) is modified, not requiring all distances to be used. The objective of this model is to maximize the number of distances that can be distributed among $t + 1$ points.

This model can be utilized with Algorithm 3, by noting that finding a feasible solution for the MDSP is equivalent to finding an optimal solution to Model (4-10) with a value equal to the size of the input. If the cardinality of the input is k and the optimal solution of (4-10) is less than k, then we can conclude that it is impossible to distribute that many distances among $t + 1$

points. Otherwise, if the value of (4-10) is k (notice that it can't be higher due to the second set of constraint), then the set of variables p_i is a feasible solution for the MDSP, but there might be a smaller solution. This leads to a binary search based algorithm, which is also described by Algorithm 3.

Algorithm 3 can work with both models presented in this section. The major difference is that model (4-10) will never be infeasible, as setting all $x_{d,i,j}$ to zero and all $p_i = i$ is always a feasible solution. Therefore, the ub is set to mid only if the optimal solution of the current model has value k and a smaller solution needs to be found or proven to be nonexistent. If the optimal solution is value is less than k, then lb is set to $mid + 1$. This goes on until a solution is proven to be optimal, when $lb > ub$, finally outputting it.

4.2 Conclusions

This chapter has introduced a polynomially sized IP model for the MDSP. While this is a major improvement over the quadratic model with pseudo-polynomial size, Model (4-8) can still achieve large proportions quickly, as its size increases with complexity $\mathbf{O}(k^3)$. To deal with this, two simpler models were also introduced: Models (4-9) and (4-10) are built assuming that the optimal solution will have a certain number of points. Then, for both of these models, it is necessary to perform a binary search on the quantity of points an optimal solution for the MDSP can have, trading some model complexity for solving these models $\Theta(\log k)$ times.

Some problems we have identified include:

- How well does a branch and bound approach works with model (4-8)? Can we find good cuts or other approaches to solve it faster?
- Symmetry seems to play a large role on the difficulty of solving all these models. Can these symmetries be broken?
- Heuristically identifying better lower and upper bounds for a given MDSP instance could dramatically improve the efficiency of Algorithm 3 using Models (4-9) and (4-10).

5
Computational Experiments and Analysis

In this chapter, we will discuss all computational experiments regarding Chapters 3 and 4. All algorithms described in this chapter have been implemented in Python 3.4. All integer and quadratic programming models are solved using Gurobi 6 Python's API, with default settings. The Numpy Python library, version 1.9.2, was used for all linear algebra computations. All experiments ran in a server with an Intel i7-3960X 3.3GHz, running Windows 7 64-bits with 64 GB of RAM.

For all experiments that utilizes Gurobi, running with default settings means that up to six threads were used when solving mathematical programs. Python was chosen as the implementation language for its rapid prototyping, specially since most computation time was going to be spent solving models with Gurobi.

Two different experiments will be presented in this chapter. The first experiment consists in testing the direct computation of variable bounds for Model (3-2) as described in Section 3.4. The second experiment is the comparison of the following solution methods for the MDSP: the linearized quadratic model (3-3), the integer programming model (4-8), Algorithm 3 using Model (4-9) and Algorithm 3 using Model (4-10).

5.1
Instances

Four different types of instances were randomly generated to conduct these tests. Most of the instances are generated by creating a set of points in a line (with a maximum distance between consecutive points), calculating all pairwise distances and then making appropriate modifications. Instance sizes were chosen in such a way to have easy, medium and hard instances. Easy instances have approximately 10 distances, medium instances have approximately 50 distances and hard instances have approximately 110 distances.

A total of 105 instances were generated to be used among all tests. Each instance was named following the convention *type-constants-identifier*, where the constants are the numbers used to generate the instance, separated by dashes, and the identifier is a 5 letter random string.

The instances are defined as follows:

– Full instances: these are PDP instances, that is, instances of size $k = \frac{n(n-1)}{2}$ for some $n \in \mathbb{N}$, with optimal solution P of cardinality n. They are complete instances, in the sense that no distances are missing. These instances were generated by sampling $n-1$ integers between 1 and m as the distances between consecutive points on the line, and calculating all pairwise distances. Five of these instances were generated for each pair between $n = 5, 10, 15$ and $m = 15, 30$, for a total of 30 instances. These instances are named following the pattern "full-n-m-identifier".

– Missing distance instances: these are MDSP instances which are not PDP instances, that is, instances of size between $k_1 = \frac{n(n-1)}{2}$ and $k_2 = \frac{n(n+1)}{2}$, with optimal solution P of cardinality greater than n. They are generated by randomly removing an appropriate number of distances from a Full instance generated from $n+1$ points. Given n, the number of distances to be removed is calculated as $l = k_2 + (k_1 + k_2)/2$, totalling a size of $\frac{n(n+1)}{2} - l$. Five of these instances were generated for each pair between $n = 5, 10, 16$ and $m = 15, 30$ (for the underlying Full instance), for a total of 30 instances. These instances are named following the pattern "miss-$(n+1)$-m-l-identifier".

– Joint instances: these are the concatenation of two Full instances generated from pair of integers (n_1, m_1) and (n_2, m_2). There is no guarantee on how many points the solution will have. Three of these instances were generated for each pair between $(n_1, n_2) = (5, 5), (10, 5), (15, 5)$ and $(m_1, m_2) = (15, 15), (30, 30)$, for a total of 18 instances. These instances are named following the pattern "joint-n_1-m_1-n_2-m_2-identifier".

– Random instances: these are instances consisting of k integers uniformly sampled in $[1, d]$. The distances d were chosen as approximations to the maximum distances found in the previous distances. Three instances were generated for each pair between $k = 10, 50, 112$ and $d = 75, 110, 200$ for a total of 27 instances. These instances are named following the pattern "drand-k-d-identifier".

5.2 Quadratic Formulation and Pruning Technique

The first computational experiment aimed at investigating the viability of the approach based on quadratic programming and variable pruning, described in Section 3.4.

The following conventions have been used for the tests:

– The pruning technique for unconstrained quadratic programs was applied to each instance with the method explained in Section 3.4. Notice that

variables were not filtered out using Algorithm 2 before applying either the lagrangian relaxation or the pruning technique, as it did not filter many variables for the generated instances.

– The subgradient optimization was initialized with all lagrangian multipliers equal to zero and the step factor was initialized as $\pi = 2$, being halved every 50 successful iterations of the subgradient optimization. The pruning technique used the trivial upper bound of each instance for its calculations.

– For each test, the subgradient optimization of the lagrangian relaxation was given a time limit of 60 seconds, triggering termination only when time ran out. The pruning technique was given another 60 seconds to run before the method was terminated.

Table 5.1 shows the results from this experiments for the 35 instances that did not time out. The pruning technique did not terminate for 70 out of 105 instances. This happened because the number of variables in Model (3-2) is equal to the sum of the distances in each instance, and thus, pseudo-polynomial on the size of the instance. For example, one of the largest random instances, containing 112 distances, leads to 11500 variables, which means that the quadratic form in the pruning technique is a 11500×11500 matrix.

As it can be seen, the results were negative, in the sense that no variable fixing information could be derived on these instances. Notice that column lb_1, which is a lower bound to Model (3-2), does not reach the trivial lower bound of $\left\lceil \frac{1}{2} + \sqrt{\frac{1}{4} + 2k} \right\rceil$. No information about which variables must always be set to 0 or to 1 was acquired.

Applying the pruning technique of (Fleischman, 2010) through the indirection of a lagrangian relaxation did not provide, during our experiments, any useful information about the MDSP. However, it is possible that this technique may still yield results for the MDSP on different (possibly smaller) instances or by proposing an improved version of Algorithm 2.

Several techniques based on the quadratic programming formulation were not successful. As it can be seen in the next section, the linearization-reformulation of the quadratic programming formulation also presents a low performance. Because of this, it was necessary to reconsider our mathematical models, leading to the integer programming formulations introduced in Chapter 4.

Table 5.1: Pruning technique results for instances that didn't time out. In this table, $LB0$ and $LB1$ are lower bounds for the size of the sets $\{x_i \mid x_i = 0\}$ and $\{x_i \mid x_i = 1\}$, respectively, while $Set0$ and $Set1$ are the number of x_i variables that were determined to be equal to 0 and 1, respectively.

name	$LB0$	$LB1$	$Set0$	$Set1$
drand-10-110-IAeNk	-969.74	-21.34	0	0
drand-10-110-RVUNe	-2342.33	-6.66	0	0
drand-10-110-xGWpl	-730.18	-24.96	0	0
drand-10-200-NRczF	-2797.45	-16.65	0	0
drand-10-200-RADUL	-2865.93	-16.88	0	0
drand-10-200-uJUJU	-1021.58	-16.35	0	0
drand-10-75-FrySA	-904.18	-26.88	0	0
drand-10-75-LMchp	-2376.37	-7.96	0	0
drand-10-75-duABx	-5154.08	-3.05	0	0
full-5-15-Bmagw	-3470.59	-3.36	0	0
full-5-15-DXuOB	-3003.78	-1.8	0	0
full-5-15-IGyKD	-115.21	-39.22	0	0
full-5-15-YPTpx	-75.24	-37.67	0	0
full-5-15-gzXps	-1843.82	-1.5	0	0
full-5-30-HemAR	-1631.46	-9.17	0	0
full-5-30-JXNyw	-667.81	-26.42	0	0
full-5-30-RgeCt	-3018.98	-8.84	0	0
full-5-30-jNdcc	-348.78	-31.39	0	0
full-5-30-mmVNa	-1866.51	-10.34	0	0
joint-5-15-5-15-JCqgc	-26066.87	-5.78	0	0
joint-5-15-5-15-cRuVK	-9790.06	-29.92	0	0
joint-5-15-5-15-kpzbA	-5848.87	-11.45	0	0
joint-5-30-5-30-JbgQE	-14163.74	-19.3	0	0
joint-5-30-5-30-UdbEW	-3213.40	-67.1	0	0
joint-5-30-5-30-yxDLl	-18605.39	-20.17	0	0
miss-6-15-2-CQUqA	-634.73	-50.41	0	0
miss-6-15-2-Jxirs	-339.47	-65.17	0	0
miss-6-15-2-PRmUu	-2078.43	-13.58	0	0
miss-6-15-2-PZdYO	-292.28	-55.64	0	0
miss-6-15-2-foYyI	-259.19	-62.36	0	0
miss-6-30-2-UiSdF	-1285.40	-34.12	0	0
miss-6-30-2-epXBf	-3935.16	-16.7	0	0
miss-6-30-2-fwQWS	-5544.22	-12.12	0	0
miss-6-30-2-mGmmu	-868.66	-42.11	0	0
miss-6-30-2-wBcAX	-4653.33	-10.36	0	0

5.3 Comparison of MDSP methods

The goal of this section is to compare different mathematical programming formulations for the MDSP, based on their ability to solve the proposed instances to completion. The MDSP mathematical programming formulations compared in this computational experiment are: the linearization-reformulation of the quadratic programming Model (3-3) with filtered variables, the integer programming Model (4-8), Algorithm 3 using Model (4-9) and Algorithm 3 using Model (4-10). For the sake of brevity, these methods will be called RLT, IP, FEAS and DISTMAX, respectively. Each method was given 3600 seconds to run each instance a single time.

All methods, except FEAS, were modified in order to utilize the information about lower and upper bounds that can be calculated before hand in the form of lower and upper bound constraints. This means that for each method, two constraints were added: a constraint such that the objective value must be equal or greater than the known lower bound and a constraint such that the objective value must be equal or lesser than the known upper bound.

The results for each type of instance can be seen in Tables 5.2, 5.3, 5.4 and 5.5. Note that these tables do not total 105 instances, but rather 95 instances. This is due to an undiagnosed and unresolved issue regarding Gurobi, in which root nodes would stall for hours, ignoring any time limits as they are checked only when a node is closed. Therefore, instances for which all methods stalled were discarded, while instances in which only some methods stalled have entries with dashes. Entries have a value of "OOM" if an out of memory exception was raised while executing it.

As expected, the linearization-reformulation of the quadratic programming model (3-3) (named RLT in the tables), is the method with the worst performance among the four proposed methods. Out of the 95 instances, it only managed to close 14 instances, 10 belonging to the full instance type and 4 belonging to the missing distance instance type. Out of these solved instances, its fastest solution time was 1.23 seconds for the instance "full-5-15-DXuOB", which took the next slowest method, IP, only 0.13 seconds to solve.

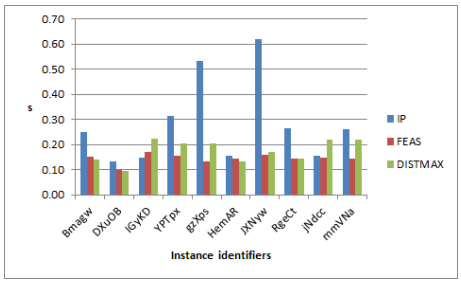
Even with lower and upper bound constraints, the RLT method fails to close other easy instances in which the other methods succeeded. For many larger instances, it even fails to find lower and upper bounds. It is also the only method to have raised out of memory exceptions, on a machine with 64 GB of RAM.

The integer programming Model (4-8) is in all cases a massive improvement over the RLT method, presenting speed-ups of CPU time ranging from 9

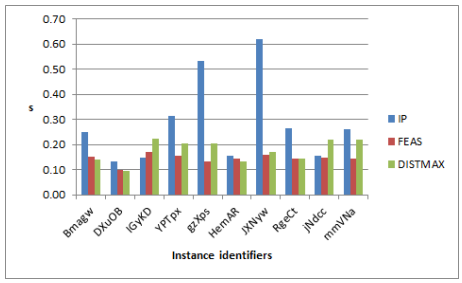
to 898 times for instances which both methods succeeded in finding an optimal solution and managing to close 32 out of the 35 easiest instances. This method could be quickly improved by providing an initial trivial solution to the integer model, as some of the larger instances fail to report an upper bound, meaning that it failed to find any feasible solution.

Figure 5.1: Time comparison for instances that IP, FEAS and DISTMAX solved. Note that the bars always follow this order. Each entry in the horizontal axis is a different instance, denoted by its unique identifier. The vertical axis measures the CPU run time in seconds.

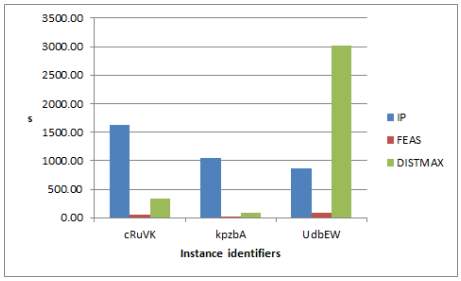
5.1(a): Full instances.



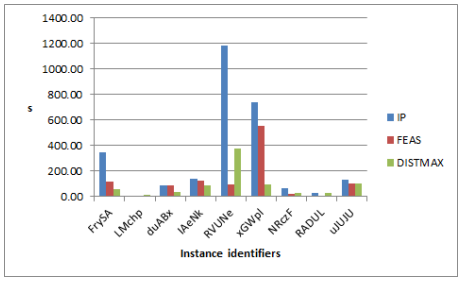
5.1(b): Missing distance instances.



5.1(c): Joint instances.



5.1(d): Random distance instances.



The methods based on binary search, FEAS and DISTMAX, have shown the best results overall, in terms of number of instances solved, CPU time for those instances and gap size for unsolved instances. Figure 5.3 shows a chart for each instance type, comparing the CPU time for all instances which IP, FEAS and DISTMAX solved. As it can be seen in Figure 5.3 and Tables 5.2, 5.3, 5.4 and 5.5, the IP formulation was consistently beat in CPU time by at least one of both FEAS and DISTMAX.

Finally, the FEAS method beats the DISTMAX method in all categories, except number of instances solved, as DISTMAX solved 34 instances and FEAS solved 33. The first method has shown better CPU time in 22 of the instances, while DISMAX showed better CPU time in 12 instances. The same occurs with the quality of upper bounds achieved: FEAS found better upper bound 17 times, while DISTMAX found the best upper bound 8 times, across all instances.

The generated instances have proven to be harder to solve than expected,

as no algorithm succeeded in solving more than 35 out of 105 proposed instances. It's also interesting to note these methods often found different solutions to the instances. For example, consider the instance "drand-10-75-duABx", with $D = \{16, 31, 40, 57, 57, 61, 65, 69, 69, 75\}$. The IP method found the optimal solution $P_{IP} = \{0, 4, 12, 16, 69, 73, 113, 144\}$, the FEAS method found the optimal solution $P_{FEAS} = \{0, 6, 14, 18, 31, 71, 75, 87\}$, the DISTMAX method found the optimal solution $P_{DISTMAX} = \{0, 2, 40, 59, 63, 71, 75, 128\}$ and the RLT method found the non-optimal solution $P_{RLT} = \{0, 151, 208, 265, 273, 297, 303, 313, 334, 372\}$.

5.4 Conclusions

Two different computational experiments were presented in this chapter. The first experiment, consisting of applying the pruning technique to the lagrangian relaxation of the quadratic programming method, failed to provide new information about the instances. This suggests that applying the pruning technique through a layer of indirection might not provide quality results overall. However, this technique might still work if Algorithm 2, or some other algorithm with the same purpose, is improved.

The second experiment consisted of comparing the linearization-reformulation of the quadratic programming Model (3-3) with filtered variables, the integer programming Model (4-8), Algorithm 3 using Model (4-9) and Algorithm 3 using Model (4-10). The first model does not show promising results, corroborating our expectations, as it has a pseudo-polynomial size on the size of the input. The second model proved to be a major improvement over the quadratic model, performing better in all metrics. Finally, the binary search based methods have shown the best results, solving more instances, achieving better times for instances that integer formulation solved and solving more instances, with the method based on Model (4-9) showing the best overall results.

Table 5.2: Results for methods RLT, IP, FEAS and DISTMAX, when applied to full instances. Column *run time* is the CPU time in seconds and columns *lb* and *ub* are the reported lower and upper bounds.

	RLT			IP			FEAS			DISTMAX		
name	run time	lb	ub	run time	lb	ub	run time	lb	ub	run time	lb	ub
full-5-15-Bmagw	3.67	5	5	0.25	5	5	0.15	5	5	0.14	5	5
full-5-15-DXuOB	1.23	5	5	0.13	5	5	0.10	5	5	0.10	5	5
full-5-15-IGyKD	45.46	5	5	0.15	5	5	0.17	5	5	0.22	5	5
full-5-15-YPTpx	5.04	5	5	0.31	5	5	0.16	5	5	0.20	5	5
full-5-15-gzXps	10.93	5	5	0.53	5	5	0.13	5	5	0.21	5	5
full-5-30-HemAR	6.73	5	5	0.16	5	5	0.14	5	5	0.13	5	5
full-5-30-JXNyw	77.39	5	5	0.62	5	5	0.16	5	5	0.17	5	5
full-5-30-RgeCt	18.45	5	5	0.27	5	5	0.14	5	5	0.15	5	5
full-5-30-jNdcc	133.68	5	5	0.15	5	5	0.15	5	5	0.22	5	5
full-5-30-mmVNa	9.95	5	5	0.26	5	5	0.14	5	5	0.22	5	5
full-10-15-NTugs	3600.00	10	23	3600.00	10	41	3600.00	10	18	3599.79	10	18
full-10-15-XvOOt	3600.00	10	25	3600.00	10	44	3599.79	10	18	3600.00	10	18
full-10-15-ZFfzV	3600.00	10	17	3600.00	10	19	3599.78	10	13	3600.00	10	13
full-10-15-cJWZy	3600.00	10	23	3600.00	10	19	3599.79	10	13	3599.81	10	18
full-10-15-zCRcz	3600.00	10	20	3600.00	10	33	3599.91	10	13	3600.00	10	18
full-10-30-EmIgJ	3600.00	10	22	3600.00	10	40	3600.00	10	18	3600.00	10	18
full-10-30-KPjnZ	3600.00	1	$+\infty$	3600.00	10	44	3600.00	10	18	3600.00	10	18
full-10-30-KyYWc	3600.00	$-\infty$	22	3600.00	10	36	3599.84	10	18	3599.78	10	18
full-10-30-eCmoy	3600.00	1	$+\infty$	3600.00	10	43	3600.00	10	18	3600.00	10	18
full-10-30-mvMQz	3600.00	$-\infty$	23	3600.00	10	44	3600.00	10	18	3600.00	10	28
full-15-15-EVTNs	3600.00	$-\infty$	28	3600.00	15	$+\infty$	3600.00	15	37	3600.00	15	25
full-15-15-adhAt	3600.00	$-\infty$	28	3600.00	15	$+\infty$	3600.00	15	25	3600.00	15	37
full-15-15-jdZur	3600.00	$-\infty$	32	3600.00	15	$+\infty$	3600.00	15	37	3600.00	15	37
full-15-15-sHNZQ	3600.00	1	$+\infty$	3600.00	15	$+\infty$	3600.00	15	37	3600.00	15	37
full-15-15-yipTk	OOM	OOM	OOM	3600.00	15	$+\infty$	3599.89	15	37	3600.00	15	60

Table 5.3: Results for methods RLT, IP, FEAS and DISTMAX, when applied to missing distance instances. Column *run time* is the CPU time in seconds and columns *lb* and *ub* are the reported lower and upper bounds.

	RLT			IP			FEAS			DISTMAX		
name	run time	lb	ub	run time	lb	ub	run time	lb	ub	run time	lb	ub
miss-6-15-2-CQUqA	3600.00	6	7	1.32	6	6	0.49	6	6	0.94	6	6
miss-6-15-2-Jxirs	3600.00	6	7	2.03	6	6	0.99	6	6	1.06	6	6
miss-6-15-2-PRmUu	30.35	6	6	0.68	6	6	0.49	6	6	0.89	6	6
miss-6-15-2-PZdYO	3600.00	6	7	1.63	6	6	0.90	6	6	0.45	6	6
miss-6-15-2-foYyI	425.49	6	6	1.80	6	6	1.06	6	6	1.15	6	6
miss-6-30-2-UiSdF	3600.00	6	7	1.35	6	6	0.69	6	6	1.53	6	6
miss-6-30-2-epXBf	771	6	6	1.22	6	6	0.61	6	6	0.25	6	6
miss-6-30-2-fwQWS	3600.00	6	7	1.39	6	6	0.28	6	6	0.28	6	6
miss-6-30-2-mGmmu	3600.00	6	7	2.03	6	6	0.38	6	6	0.62	6	6
miss-6-30-2-wBcAX	230.23	6	6	1.47	6	6	0.37	6	6	3.47	6	6
miss-11-15-5-VDdhV	3600.00	11	23	3600.00	11	51	3600.00	11	20	3600.00	11	20
miss-11-15-5-awYaT	3600.00	1	$+\infty$	3600.00	11	51	3600.00	11	20	3600.00	11	20
miss-11-15-5-bERIo	3600.00	11	23	3600.00	11	50	3600.00	11	20	3599.82	11	20
miss-11-15-5-dmzVR	3600.00	11	20	3600.00	11	49	3600.00	11	15	3599.83	11	20
miss-11-15-5-yVhpG	3600.00	1	$+\infty$	3600.00	11	51	3599.95	11	15	3600.00	11	15
miss-11-30-5-BgZoM	3600.00	$-\infty$	$+\infty$	3600.00	11	50	3599.86	11	20	3600.00	11	31
miss-11-30-5-UimPG	3600.00	1	$+\infty$	3600.00	11	46	3599.97	11	20	3600.00	11	20
miss-11-30-5-nMlvm	3600.00	$-\infty$	24	3600.00	11	$+\infty$	3600.00	11	20	3600.00	11	31
miss-11-30-5-pYCsc	3600.00	1	$+\infty$	3600.00	11	51	3600.00	11	20	3600.00	11	20
miss-11-30-5-qsAHF	3600.00	1	$+\infty$	3600.00	11	49	3600.00	11	20	3600.00	11	20
miss-17-15-8-FYRet	3600.00	$-\infty$	$+\infty$	3600.00	17	$+\infty$	3600.00	17	44	3600.00	17	129
miss-17-15-8-JroTc	3600.00	$-\infty$	$+\infty$	3600.00	17	$+\infty$	3600.00	17	129	3600.00	17	129
miss-17-15-8-qzzLS	3600.00	$-\infty$	$+\infty$	3600.00	17	$+\infty$	3600.00	17	129	3600.00	17	129
miss-17-15-8-vRtHH	3600.00	$-\infty$	$+\infty$	3600.00	17	$+\infty$	3600.00	17	44	3600.00	17	129
miss-17-15-8-zcQEk	3600.00	$-\infty$	$+\infty$	3600.00	17	$+\infty$	3600.00	17	129	3600.00	17	44
miss-17-30-8-FXPyU	OOM	OOM	OOM	3600.00	17	$+\infty$	3600.00	17	129	3600.00	17	73
miss-17-30-8-dtaZd	OOM	OOM	OOM	3600.00	17	$+\infty$	3600.00	17	129	3600.00	17	129

Table 5.4: Results for methods RLT, IP, FEAS and DISTMAX, when applied to joint instances. Column *run time* is the CPU time in seconds and columns *lb* and *ub* are the reported lower and upper bounds.

	RLT			IP			FEAS			DISTMAX		
name	run time	lb	ub	run time	lb	ub	run time	lb	ub	run time	lb	ub
joint-5-15-5-15-JCqgc	3600.00	7	10	3600.00	7	9	976.78	9	9	917.82	9	9
joint-5-15-5-15-cRuVK	3600.00	7	10	1630.22	8	8	60.95	8	8	330.54	8	8
joint-5-15-5-15-kpzbA	3600.00	7	9	1041.94	8	8	10.37	8	8	88.19	8	8
joint-5-30-5-30-JbgQE	3600.00	7	12	3599.86	7	9	3600.00	7	10	3600.00	7	10
joint-5-30-5-30-UdbEW	3600.00	7	11	873.44	8	8	82.75	8	8	3018.05	8	8
joint-5-30-5-30-yxDLl	3600.00	7	11	3599.89	7	9	3600.00	7	10	1849.37	9	9
joint-10-15-5-15-ERzSf	3600.00	$-\infty$	22	3600.00	11	53	3600.00	11	21	3600.00	11	21
joint-10-15-5-15-VDLUB	3600.00	11	23	3600.00	11	36	3600.00	11	15	3600.00	11	21
joint-10-15-5-15-vxEov	3600.00	11	27	3600.00	11	29	3599.80	11	15	3600.00	11	21
joint-10-30-5-30-CrgyC	3600.00	$-\infty$	25	3600.00	11	37	3600.00	11	21	3599.85	11	21
joint-10-30-5-30-EAlFt	3600.00	1	$+\infty$	3600.00	11	54	3600.00	11	21	3600.00	11	21
joint-10-30-5-30-rpeSd	3600.00	1	$+\infty$	3600.00	11	53	3600.00	11	21	3600.00	11	21
joint-15-15-5-15-LDaWI	3600.00	$-\infty$	$+\infty$	3600.00	16	$+\infty$	3600.00	16	116	3600.00	16	40
joint-15-15-5-15-WwbAA	3600.00	$-\infty$	32	3600.00	16	$+\infty$	3600.00	16	116	3600.00	16	40
joint-15-15-5-15-iQFaV	3600.00	$-\infty$	$+\infty$	3600.00	16	$+\infty$	3600.00	16	40	3600.00	16	66
joint-15-30-5-30-YTGPP	OOM	OOM	OOM	3600.00	16	$+\infty$	3600.00	16	116	3600.00	16	116
joint-15-30-5-30-cUZHK	OOM	OOM	OOM	3600.00	16	$+\infty$	3600.00	16	116	3600.00	16	116
joint-15-30-5-30-dwnXW	3600.00	$-\infty$	$+\infty$	3600.00	16	$+\infty$	3600.00	16	116	3600.00	16	116

Table 5.5: Results for methods RLT, IP, FEAS and DISTMAX, when applied to random distances instances. Column *run time* is the CPU time in seconds and columns *lb* and *ub* are the reported lower and upper bounds.

	RLT			IP			FEAS			DISTMAX		
name	run time	lb	ub	run time	lb	ub	run time	lb	ub	run time	lb	ub
drand-10-75-FrySA	3600.00	5	8	343.17	8	8	117.98	8	8	59.85	8	8
drand-10-75-LMchp	3600.00	5	8	3.70	7	7	2.33	7	7	12.17	7	7
drand-10-75-duABx	3600.00	5	10	86.04	8	8	83.97	8	8	33.39	8	8
drand-10-110-IAeNk	3600.00	5	9	140.80	8	8	125.21	8	8	88.11	8	8
drand-10-110-RVUNe	3600.00	5	9	1185.35	8	8	98	8	8	373.82	8	8
drand-10-110-xGWpl	3600.00	4	10	740.38	8	8	552.61	8	8	92.63	8	8
drand-10-200-NRczF	3600.00	5	9	63.01	7	7	17.14	7	7	30.19	7	7
drand-10-200-RADUL	3600.00	5	9	26.20	7	7	4.15	7	7	30.66	7	7
drand-10-200-uJUJU	3600.00	5	10	135.09	8	8	103.51	8	8	100.20	8	8
drand-50-75-GrPYu	3600.00	11	23	3600.00	11	47	3599.99	11	20	3600.00	11	20
drand-50-75-MQpCM	3600.00	1	$+\infty$	3600.00	11	37	3600.00	11	20	3600.00	11	20
drand-50-75-kDuRR	3600.00	11	24	3600.00	11	31	3600.00	11	20	3600.00	11	31
drand-50-110-qwGbF	3600.00	1	$+\infty$	3600.00	11	48	3599.85	11	31	3600.00	11	31
drand-50-110-rzTGD	3600.00	$-\infty$	$+\infty$	3600.00	11	51	3600.00	11	31	3599.84	11	20
drand-50-110-zeXut	3600.00	1	$+\infty$	3600.00	11	23	3599.82	11	20	3599.91	11	31
drand-50-200-GDWds	3600.00	$-\infty$	$+\infty$	3600.00	11	46	3600.00	11	31	3600.00	11	31
drand-50-200-oIURA	3600.00	$-\infty$	-1	3600.00	11	51	3600.00	11	31	3600.00	11	31
drand-112-75-HKvAw	3600.00	$-\infty$	32	3600.00	16	$+\infty$	1452	16	27	3600.00	16	39
drand-112-75-mnjBD	-	-	-	3600.00	16	$+\infty$	3060	16	39	3600.00	16	39
drand-112-110-bBfWZ	3600.00	$-\infty$	$+\infty$	3600.00	16	$+\infty$	3600.00	16	64	3600.00	16	64
drand-112-110-qiBfT	3600.00	$-\infty$	$+\infty$	3600.00	16	$+\infty$	3600.00	16	113	3600.00	16	39
drand-112-110-zMxaB	3600.00	$-\infty$	40	3600.00	16	$+\infty$	3600.00	16	64	3600.00	16	64
drand-112-200-GOsUT	OOM	OOM	OOM	3600.00	16	$+\infty$	3600.00	16	113	3600.00	16	113
drand-112-200-WdgfA	OOM	OOM	OOM	3600.00	16	$+\infty$	3600.00	16	113	3600.00	16	113
drand-112-200-zzVSk	OOM	OOM	OOM	-	-	-	3600.00	16	64	-	-	-

6
Conclusions and Future Works

In this thesis, we have proposed new mathematical programming formulations for the min distance superset problem. The first was a quadratically constrained $0-1$ program (3-2) and the second an integer program (4-8). For the first, it was shown how to apply the pruning technique for quadratic unconstrained programs, proposed in (Fleischman, 2010), via a lagrangian relaxation. Unfortunately, the pruning technique did not contribute to reduce the size of the problems via variable fixing, showing poor experimental performance when dealing with a variety of MDSP instances. For the second model, two solution approaches involving binary searches on the number of points in a solution were introduced. These proved to be successful, showing improved results when compared to the integer and reformulation-relinearization of que quadratically constrained $0-1$ models.

Future works include:

- Studying the approximability of the MDSP. Such a result may exist, as Cieliebak has proven it for a similar problem (Cieliebak et al., 2003).
- Study the symmetry in MDSP solutions. Symmetry breaking constraints may improve the efficiency of all models considered in this thesis, by significantly reducing the search space.
- Improve Algorithm 2 or find a better substitute. Reducing the number of variables in the quadratic $0-1$ model will drastically improve its efficiency, specially if it can be reduced to a polynomial size on the input.
- Apply and study branch and bound techniques for Model (4-8).
- Develop heuristics which can improve the trivial lower bounds for a given MDSP instance. This should improve the efficiency of all presented models, specially the binary search based methods.

7
Bibliography

ANSTREICHER, K. M. Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. **Journal of Global Optimization**, Springer, vol. 43, no. 2-3, p. 471–484, 2009. 3.1.2

CIELIEBAK, M. **Algorithms and hardness results for DNA physical mapping, protein identification, and related combinatorial problems**. Tese (Doutorado) — ETH Zurich, 2003. 2.2

CIELIEBAK, M.; EIDENBENZ, S.; PENNA, P. **Noisy data make the partial digest problem NP-hard**. [S.l.]: Springer, 2003. 1, 3, 1, 2.2, 2.2, 6

DAKIC, T. **On the turnpike problem**. Tese (Doutorado) — Simon Fraser University, 2000. 1, 2.1, 3

DAURAT, A.; GÉRARD, Y.; NIVAT, M. The chords problem. **Theoretical Computer Science**, Elsevier, vol. 282, no. 2, p. 319–336, 2002. 1, 2.1

DAURAT, A.; GÉRARD, Y.; NIVAT, M. Some necessary clarifications about the chords problem and the partial digest problem. **Theoretical computer science**, Elsevier, vol. 347, no. 1, p. 432–436, 2005. 1, 2.1

FLEISCHMAN, D. **An Improved Exact Method for the UBQP**. Tese (Doutorado) — PUC-Rio, 2010. 1, 3.1.1, 3.2, 3.3, 3.3, 3.5, 5.2, 6

LEMKE, P.; WERMAN, M. On the complexity of inverting the autocorrelation function of a finite integer sequence, and the problem of locating n points on a line, given the (n 2) unlabelled distances between them. **Preprint**, vol. 453, 1988. 1, 2.1

NADIMI, R.; FATHABADI, H. S.; GANJTABESH, M. A fast algorithm for the partial digest problem. **Japan journal of industrial and applied mathematics**, Springer, vol. 28, no. 2, p. 315–325, 2011. 1, 2.1

ROSENBLATT, J.; SEYMOUR, P. D. The structure of homometric sets. **SIAM Journal on Algebraic Discrete Methods**, SIAM, vol. 3, no. 3, p. 343–350, 1982. 1, 2.1

SHERALI, H. D.; ADAMS, W. P. **A reformulation-linearization technique for solving discrete and continuous nonconvex problems**. [S.l.]: Springer Science & Business Media, 2013. 3.1.2

SKIENA, S. S.; SMITH, W. D.; LEMKE, P. Reconstructing sets from interpoint distances. In ACM. **Proceedings of the sixth annual symposium on Computational geometry**. [S.l.], 1990. p. 332–339. 1, 3, 1, 2.1, 2.1

SKIENA, S. S.; SUNDARAM, G. A partial digest approach to restriction site mapping. **Bulletin of Mathematical Biology**, Springer, vol. 56, no. 2, p. 275–294, 1994. 1

WOEGINGER, G. J.; YU, Z. On the equal-subset-sum problem. **Information Processing Letters**, Elsevier, vol. 42, no. 6, p. 299–302, 1992. 1

ZHANG, Z. An exponential example for a partial digest mapping algorithm. **Journal of Computational Biology**, vol. 1, no. 3, p. 235–239, 1994. 1, 1, 2.1, 2.1