NTNU – Trondheim
Norwegian University of
Science and Technology

Lagrangian Relaxations for the Prize-Collecting Steiner Tree Problem

Lars Dahle

Master of Science in Physics and Mathematics
Submission date: May 2013
Supervisor: Anne Kværnø, MATH
Co-supervisor: Truls Flatberg, Sintef Trondheim

Norwegian University of Science and Technology
Department of Mathematical Sciences

Acknowledgments

This thesis was written during the spring of 2013, as a Master Thesis at Industrial Mathematics NTNU. I would like to thank Truls Flatberg for his guidance throughout this work, Anne Kværnø for her supervision with this paper, and Arnar Flatberg and Vidar Beisvåg for data and guidance for the biology chapter.

Trondheim, May 2013
Lars Dahle

Abstract

In this thesis we look at the NP-hard Prize-Collecting Steiner Tree Problem (PCSTP). We show an application within biology, where the PCSTP is used to identify coregulated genes which are differently expressed for diffuse stomach cancer with and without signet ring cells. Identifying these genes can help to create better treatment for stomach cancer.

We use Lagrangian relaxation to solve the PCSTP. Two relaxations are tested, one creating tights bounds and one weak bounds. For the tightest relaxation we test two heuristic methods for creating primal bounds, and for the unrooted PCSTP we find the optimal solution for all but one of the instances with both heuristics.

A multiple rooted generalization of the PCSTP is introduced, and shown to be easily handled by the formulations. We do not obtain as tight bounds as for the corresponding unrooted instances.

Sammendrag

I denne oppgaven ser vi på det NP-harde Prize-Collecting Steiner Tree Problem (PCSTP). Vi viser en anvendelse innen biologi, hvor PCSTP blir brukt for å identifisere koregulerte gener som er forskjellig uttrykt for diffus magekreft med og uten signet ring celler. Å identifisere disse genene kan bidra til bedre behandling av magekreft.

Vi bruker Lagrange-relaksering for å løse PCSTP. Vi har relaksert to formuleringer, hvor en gir sterke grenser og en gir svake. For den sterkeste har vi testet to heuristikker. For testproblemene uten rot finner begge heuristikkene optimal løsning for alle problemene bortsett fra ett.

Vi introduserer en generalisert versjon av PCSTP, med mulighet for å spesifisere flere røtter, og viser at dette lett lar seg løse med formuleringene. Grensene blir ikke like sterke som for de tilsvarende problemene uten rot.

Contents

Introduction

Integer programming is a field concerned with identifying the optimal solution from a set of possible solutions. It is heavily used in finance and economics, but also in several other fields. In this thesis we show an application within biology, relevant for cancer research.

We look at the NP-hard Prize-Collecting Steiner Tree Problem (PCSTP). This problem has applications within, amongst others, design of certain types of distribution networks. We show different formulations for the PCSTP as an integer program, and use Lagrangian relaxation and heuristics to obtain bounds. For one of the formulations the bounds are shown to be strong for most of the test instances.

The formulations are tested against data from stomach cancer research. The goal is to identify genes which are expressed differently in diffuse cancer with and without signet ring cells. As a significant amount of signet ring cells are generally associated with a worse prognosis for the patient, this can potentially lead to better treatments.

In Chapter 1 we introduce the theory used in this thesis. In Chapter 2 we give a formal definition of the PCSTP and give an outline of solution methods from literature. The formulations are described in Chapters 3 - 5. In Chapter 6 we introduce the test instances used for the problem. In Chapter 7 we show how the relaxations are implemented, and in Chapter 8 we give computational results and conclusions.

Notations and Glossary

$\delta(u)$: The set of all edges adjacent to vertex u.
$\delta^+(u)$: The set of all arcs out of vertex u.
$\delta^-(u)$: The set of all arcs into vertex u.

BIP: Binary Integer Program, p. 3
GWMP: Göemans-Williamson Minimization Problem, p. 12
IP: Integer Program, p. 3
LP: Linear Program, p. 4
MIP: Mixed Integer Program, p. 3
MST: Minimal Spanning Tree, p. 9
MTZ: Miller-Tucker-Zemlin formulation for the PCSTP, p. 31
NWMP: Net Worth Maximization Problem, p. 11
PCSTP: Prize-Collecting Steiner Tree Problem, p. 11
SPP: Shortest Path Problem, p. 10

Vectors are normally denoted by lowercase letters, and matrices with uppercase letters. When not specified, vectors and matrices are assumed to be of appropriate dimensions.

$G = (V, E)$ denotes a graph, with V as the set of vertices or nodes and E as the set of edges.

Chapter 1

Integer programming

Definition: An *integer programming* (IP) problem can be stated as

$$\begin{aligned} Z = \min_x cx \\ Ax \leq b \\ x \geq 0 \\ x \in \mathbb{Z}^n \end{aligned}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. In a *mixed integer program* (MIP) some of the components of x may be real numbers. *Binary integer programming* (BIP) is often used when x is restricted to be binary.

To solve MIPs a straight forward approach of enumerating all solutions is possible, but as the number of possible solutions for MIPs grows rapidly, quicker algorithms are normally needed. A lower (upper) bound of a maximization (minimization) problem is from here on referred to as a primal bound, and an upper (lower) bound of a maximization (minimization) problem is referred to as a dual bound. The goal is to find good algorithms for the primal and dual bounds.

The *branch-and-bound* algorithm is built on a systematic way of partial enumeration of solutions, where primal and dual bounds stop the enumeration in certain regions of the solution space when the optimal solution is guaranteed not to be there. The LP-relaxation, defined in Section 1.1, is often used, but stronger relaxations may also be implemented in a branch-and-bound framework.

1.1 Dual bounds

The standard approach for finding dual bounds is to solve a relaxation of the original problem. Good strategies vary from problem to problem. Here we give a formal definition of relaxation and we give an introduction to Lagrangian Relaxation which will be used for the PCSTP from [15].

Definition: A problem (P^r) $Z^r = max\{f(x) : x \in \mathcal{P}^r \subseteq \mathbb{R}^n\}$ is a *relaxation* of a problem (P) $Z = max\{c(x) : x \in \mathcal{P} \subseteq \mathbb{R}^n\}$ if

$$\begin{aligned} \mathcal{P} &\subseteq \mathcal{P}^r \quad \text{and} \\ f(x) &\geq c(x) \quad \forall x \in \mathcal{P}. \end{aligned}$$

The reason for looking at a relaxation of a problem is that it will often be an easier problem to solve, and it will hopefully give some valuable information about the original problem. The main result here is that the optimal value of a relaxation is at least as good as the optimal value of the original problem, formally stated in the next proposition.

Proposition 1.1.1 *If P^r is a relaxation of P, then $Z^r \geq Z$.*

Proof: Let x be an optimal solution of P. Then $x \in \mathcal{P} \subseteq \mathcal{P}^r$, so $x \in \mathcal{P}^r$. Since $Z = c(x) \leq f(x)$, and $f(x)$ is a lower bound for Z^r, we have $Z \leq f(x) \leq Z^r$. □

An easy relaxation of a MIP is it's *Linear Programming* (LP) relaxation. This is obtained by simply removing the restriction that solutions must be integers. Thus the relaxation is easily formulated for any given problem, and in most cases it is considered relatively easy and quick to solve, by for instance the simplex method. Because of this the LP-relaxation is frequently used in a branch-and-bound framework in general purpose MIP-solvers. It is however often possible to find stronger relaxations.

1.1.1 Lagrangian relaxation

In the 1970's it was observed that many hard integer programming problems could be viewed as easy problems complicated by a relatively small set of side constraints. Relaxing or dualizing these side constraints could thus produce easier problems, giving dual bounds on the original problem. As with the LP-relaxation, these bounds can then be implemented in a branch-and-bound algorithm to solve the original problem [3].

Given a combinatorial optimization problem formulated as an integer program,

$$\begin{aligned} & Z = \min cx \\ & Ax \leq b \\ & Dx \leq e \\ \text{(IP)} \qquad & x \in \mathbb{Z}^n_+ \end{aligned}$$

where $Ax \leq b$ are 'nice' in the sense that an integer program with just these constraints are relatively easy. Then one relaxation to (IP) is given by simply dropping the complicating constraints $Dx \leq e$, and solving the remaining problem. The resulting bounds for this is often weak, as some constraints are totally ignored, so Lagrangian relaxation give a way of penalizing breached constraints, without actually solving the entire (IP).

Let $u \geq 0$ be a vector of *Lagrange multipliers*, then a relaxation to (IP) is the *Lagrangian relaxation (subproblem) of (IP) with parameter u*

$$\begin{aligned} & Z_D(u) = \min cx + u(Dx - e) \\ & Ax \leq b \\ & u \geq 0 \\ \text{(IP(u))} \qquad & x \in \mathbb{Z}_+^n. \end{aligned}$$

If the relaxed constraints are equality constraints then u is not restricted to be non-negative, and if the constraints are on form $Dx \geq e$ then either one substitutes $u(Dx - e)$ with $u(e - Dx)$ in the objective function or require $u \leq 0$. For the following theory, we assume the first is done such that $u \geq 0$. As the feasible region of (IP(u)) is obviously at least as great as that of (IP), and

$$Z_D(u) \leq cx + u(Dx - e) \leq Z$$

for all feasible x of (IP), (IP(u)) is a relaxation of (IP) for all $u \geq 0$. Thus solving (IP(u)) gives a dual bound on the solution of (IP).

To get the best dual bound on (IP), we maximize (IP(u)), giving the *Lagrangian Dual Problem*

$$\text{(LD)} \qquad Z_{LD} = \max Z_D(u).$$

(LD) is commonly solved by a *subgradient method*, which is introduced in Section 1.1.4. In general it is not possible to guarantee that $Z_{LD} = Z$, but this frequently happens for some problems [3]. Also the solution to (LD) often resembles feasible solutions to (IP) and perturbations can create good feasible solutions, which is further explained in Section 1.2.

During the solution process of (LD), solutions which are feasible to (IP) can occur, but are not automatically guaranteed to be optimal solutions of (IP). We now state the optimality conditions for the Lagrangian relaxation method [15].

Proposition 1.1.2 : Global optimality conditions
If
(i) $x(u)$ is an optimal solution of (IP(u)),
(ii) $Dx(u) \leq e$ (feasibility of (IP)),
(iii) The element $(Dx(u) = d)_i$ whenever $u_i > 0$ (complementarity),
(iv) $u \geq 0$,
then x(u) is optimal for (IP).

1.1.2 Lagrangian decomposition

Whenever an integer program contains several 'nice' bulks of constraints, or several easy to solve subproblems, Lagrangian decomposition might be effective. Given a combinatorial optimization problem of form (IP) where $Ax \leq b$ and $Dx \leq e$ are both 'nice' when the other is not in the formulation, one may duplicate the variables

to obtain a bound at least as strong as (IP(u)) [4]. Let $y = x$, then problem (IP) can be transformed to

$$\begin{aligned} & Z' = \min cx \\ & Ax \leq b \\ & Dy \leq e \\ & y = x \\ \text{(IP')} \qquad & x, y \in \mathbb{Z}_+^n \end{aligned}$$

Then the equality of x and y can be relaxed in a Lagrangian fashion, yielding a *Lagrangian Decomposition of (IP')*

$$\begin{aligned} & Z'_D(u) = \min cx - u(y - x) \\ & Ax \leq b \\ & Dy \leq e \\ \text{(IP'(u))} \qquad & x, y \in \mathbb{Z}_+^n, u \in \mathbb{R}^n \end{aligned}$$

This give the Lagrangian dual problem,

$$\text{(LD')} \qquad Z'_{LD} = \max Z'_D(u).$$

In the subgradient method, the solution process for Lagrangian decomposition is similar to the one of Lagrangian relaxation, with the modification that both subproblems are solved at each iteration.

1.1.3 Strength of relaxation methods

We here state the theory of when the Lagrangian relaxation give a stronger bound than the LP-relaxation, and when Lagrangian decomposition give a stronger bound than Lagrangian relaxation. The theory is based on [4, 6], and will later be used for a discussion of the PCSTP. Proofs for both lemmas can be found in [4].

It is known that the feasible region of an IP can be replaced with its convex hull. A partially convexified relaxation of (IP) can then be stated as,

$$(\tilde{IP}) \qquad \min_x \{cx : Dx \leq e, x \in conv(X), x \geq 0\}$$

where $conv(\cdot)$ denote the convex hull, and $X = \{Ax \leq b, x \in \mathbb{Z}^n\}$.

Lemma 1.1.3 *The optimal value of $(\tilde{IP})$ is equal to the optimal value of (LD).*

We notice that if $Ax \leq b$ already have the integrality property, i.e. the solution of $Ax \leq b$ is integer, then $x \in conv(X) \Leftrightarrow Ax \leq b$ in $(\tilde{IP})$ give that the Lagrangian relaxation yield the same solution as the LP-relaxation. Thus in order for the Lagrangian relaxation to possibly create a stronger bound than the LP-relaxation, the constraints $Ax \leq b$ must not already provide an 'exact' description of $conv(X)$.

This said, a Lagrangian relaxation can have other preferable qualities. Information about primal solutions is often easily available, and for some problems an exponential number of constraints may be tackled easily by a subproblem. This will be shown to be the case for the PCSTP.

For the case of Lagrangian Decomposition, we start by another partial convexified relaxation of (IP),

$$(I\tilde{P}') \qquad \min_x\{cx : x \in conv(Y) \cap conv(X)\}$$

where $conv(Y) = \{Dx \leq e, x \in \mathbb{N}^n\}$. Then we have the following result,

Lemma 1.1.4 *The optimal value of $(I\tilde{P}')$ is equal to the optimal value of (LD').*

Thus if none of the bulks of constraint give the convex hull, the Lagrangian Decomposition can give a strictly better bound than the Lagrangian relaxation of just one of the bulks. Further if one of the bulks give the convex hull, the Lagrangian Decomposition give a value equal to the best of the two Lagrangian relaxations, and if both bulks give the convex hull we get the same value as the LP-relaxation. An example of this for a generalization of the PCSTP can be found in [6], where they also generalize these results for decomposition of more than two subproblems.

1.1.4 Subgradient Method

The subgradient method is commonly used in literature to solve the Lagrangian dual problem [2, 6, 11].

Let a Lagrangian dual problem be given by $\max\{Z_D(u) : u \in \Pi\}$, where Π imposes nonnegativity restrictions on some components of the Lagrangian multiplier u. Let $Ax \leq b$ be the relaxed constraint. The subgradient algorithm can then be outlined as in Algorithm 1 [6, 12]. The form of $Z_D(u)$ is given in Figure 1.1.

Algorithm 1 Subgradient Method for (IP)

Step 0: Initialization. Set $u_0 = 0$, $l_0 \in (0, 2]$, LB $= -\inf$. Use some heuristic to find an upper bound UB.

Step 1: Solve the relaxed subproblem given u_k to get x_k and $Z_D(u_k)$. Set LB= $\max(Z_D(u_k)$,LB). If the lower bound has not increased in a set number of iterations, reset matrices to earlier point and half the step size scalar l_k.

Step 2: If x_k is feasible for the original problem, set UB $= \min(\text{UB}, Z(x_k))$. If not a heuristic can be used to find a feasible solution with value Z_{heu}, and set UB $= \min(\text{UB}, Z_{heu})$.

Step 3: Calculate the subgradient $g_k = Ax_k - b$, a search direction $d_k = g_k + \sigma_k d_{k-1}$ and a step length $t_k = l_k(\text{UB} - \text{LB})/||Ax_k - b||^2$.

Step 4: Check convergence and stopping criterions, if satisfied then stop.

Step 5: Update Lagrangian multiplier $u_{k+1} = P_\Pi(u_k + t_k d_k)$, where P_Π denotes the projection onto Π, which in this context simply amounts to setting negative-valued components of non negatively restricted variables to zero. Set $k = k + 1$ and goto Step 1.

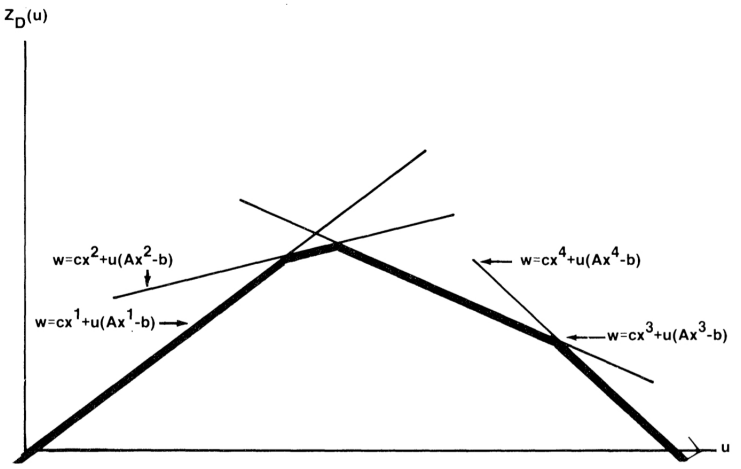

Figure 1.1: The form of $Z_D(u)$. Illustration from [3].

For Step 3 we test two different search directions from [6]. The first is the one of the classic subgradient method, where $\sigma_k = 0$ such that $d_k = g_k$. This is denoted by SG1 in the results section. The second search direction we denote by SG2, and here $\sigma_k = \frac{||g_k||}{||d_{k-1}||}$. For SG2 $d_0 = g_0$. In [6] they test nine search directions, and

SG2 is shown computationally to be the strongest for their problems (they denote it by SG6). As our formulation is based on their formulation, we have chosen to test this in this thesis.

In Step 4 we output an optimal solution if UB−LB $< 10^{-6}$, or force stopping if a predetermined number of iterations have been completed.

1.2 Primal bounds

It is possible to obtain feasible solutions to (IP) during the solution process of the dual problem. If all the relaxed constraints are equalities, this solution is optimal. This is however not guaranteed to be the case if the relaxed constraints are inequalities [3], then we need to fulfill the optimality conditions of Proposition 1.1.2. Any feasible solution may however be used to find a primal bound.

During the solution process of the dual problem we often find close to feasible solutions. Lagrangian heuristics use this information to create feasible solutions of the original problem. Most subgradient-based heuristics start with the x_k found at an iteration of the solution process and use a greedy algorithm to make it feasible. This will thus resemble a multistart to classical heuristics. A refinement of the obtained feasible solution can also be done by for instance a local search algorithm [4].

1.3 Graph Theory

In this section we state two famous problems from graph theory, state commonly used solution algorithms for these, and give MIP formulations for the problems. These will later be shown to be subproblems of the PCSTP.

Minimum Spanning Tree (MST): Given a connected, undirected edge weighted graph $G = (V, E)$. The MST problem is to find the least cost tree that spans all vertices in G. Two commonly used solution methods for the MST is Prim's and Kruskal's algorithm.

A linear programming formulation of MST, with edge costs c_{ij} and x_{ij} denoting if an edge is part of the solution, can be stated as

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{1.1a}$$

$$\sum_{(i,j)\in E} x_{ij} = |V| - 1, \tag{1.1b}$$

$$\sum_{(i,j)\in E:i,j\in S} x_{ij} \leq |S| - 1, \qquad \forall S \subset V, 2 \leq |S| \leq |V| - 2 \tag{1.1c}$$

$$0 \leq x_{ij} \leq 1, \qquad \forall (i,j) \in E. \tag{1.1d}$$

Constraint (1.1b) ensures that the solution contain exactly $|V| - 1$ edges, and constraint (1.1c) ensures the chosen edges are cycle-free. The extreme points of the polyhedron defined by (1.1) are the 0-1 incidence vectors of spanning trees [6].

Shortest Path Problem (SPP): Given a connected, edge weighted graph $G = (V, E)$, the SPP consist of finding the minimum cost path between sets of vertices. Commonly used single source algorithms are Dijkstra's algorithm whenever all edge costs are non-negative, and Bellman-Ford algorithm when some edge weights are negative. The Floyd-Warshall algorithm solves the SPP for all pairs of vertices.

Let $G = (V, A)$ be an edge weighted directed graph, with costs c_{ij} for arc (i, j). A linear programming formulation of the single source $0 \in V$ and single sink $u \in V$ SPP is

$$\min \sum_{(i,j)\in A} c_{ij} f_{ij} \tag{1.2a}$$

$$\sum_{j\in\delta^+(0)} f_{0j} - \sum_{j\in\delta^-(0)} f_{j0} = 1 \tag{1.2b}$$

$$\sum_{j\in\delta^+(u)} f_{uj} - \sum_{j\in\delta^-(u)} f_{ju} = -1 \tag{1.2c}$$

$$\sum_{j\in\delta^+(i)} f_{ij} = \sum_{j\in\delta^-(i)} f_{ji} \qquad \forall v \in V, v \neq 0, v \neq u \tag{1.2d}$$

$$0 \leq f \tag{1.2e}$$

where f_{ij} is 1 if an arc is part of the solution. Notice also that it is sufficient with either (1.2b) or (1.2c) in the formulation, as (1.2d) require balance of all vertices except the source and sink. Then with either (1.2b) or (1.2c) in the formulation, the other vertex will have to behave as source or sink.

Chapter 2

Prize-Collecting Steiner Tree Problem

The term Prize-Collecting was introduced by Balas in 1989 for the famous traveling salesman problem [1], and has since been used in various combinatorial optimization problems. The common characteristic is paying a cost for including an edge to get the prize from an adjacent vertex.

In this chapter we will define the PCSTP, give a summary of some earlier studies, and state some reduction tests which can be used for the PCSTP.

2.1 Definitions

The Prize-Collecting Steiner Tree Problem in Graphs is an NP-hard problem from graph theory. We now state two optimizationally equivalent definitions commonly used in literature.

Definition: Prize-Collecting Steiner Tree Problem in Graphs
Given a connected undirected graph $G = (V, E)$ with non-negative weights on the vertices $p(v) \quad \forall v \in V$, and non-negative weights on the edges $c(e) \quad \forall e \in E$. The linear Prize-Collecting Steiner Tree Problem in graphs is to find (either a single vertex or) a connected graph $S = (V_S, E_S) \subseteq G$, that maximizes

$$\text{profit}(S) = \sum_{v \in V_S} p(v) - \sum_{e \in E_S} c(e)$$

The PCSTP can be rooted if we require a specific vertex to be part of the solution, and unrooted if no such requirement is made. We will also look at a generalization where we require several vertices to be part of the solution, giving a multiple rooted PCSTP.

As the edges are non-negative solutions may contain cycles if and only if the cycles have zero cost, otherwise the solution is a tree. Further whenever an optimal

solution contains a cycle there will also exist trees that give the same value. These trees can be found by simply removing one of the edges in the cycle. Due to this fact we can restrict the solution to be a tree, which will be done in all of the MIP-formulations of the PCSTP.

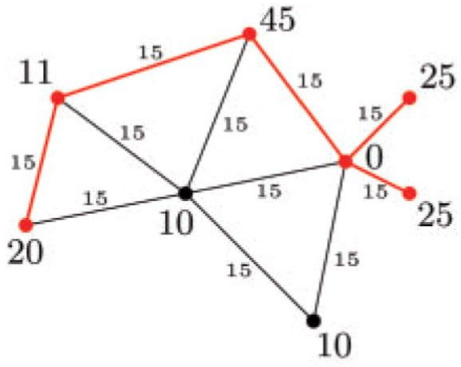

Figure 2.1: Example of a PCSTP, red line illustrates the solution tree. Illustration from [9]

The definition given above is often called the *Net Worth Maximization Problem* (NWMP) in literature, while the optimizationally equivalent *Goemans-Williamson Minimization Problem* defined below will give the same optimal solution. We will use both for arguments and explanations during this thesis.

Definition: The Goemans-Williamson Minimization Problem, (GWMP). Minimize the cost of edges plus prizes or penalties of not including a vertex in the tree

$$GW(S) = \sum_{v \notin V_S} p(v) + \sum_{e \in E_S} c(e)$$

Proposition 2.1.1 *NWMP and GWMP as stated above are equivalent with respect to optimization.*

Proof:

$$\max \sum_{v \in V_S} p(v) - \sum_{e \in E_S} c(e) = \max \sum_{v \in V} p(v) - \sum_{v \notin V_S} p(v) - \sum_{e \in E_S} c(e)$$
$$\Leftrightarrow \min - \sum_{v \in V} p(v) + \sum_{v \notin V_S} p(v) + \sum_{e \in E_S} c(e)$$

As $\sum_{v \in V} p(v)$ is constant for any given problem, NWMP and GWMP have the same optimal solution. □

The GWMP-definition is most commonly used in literature, and is the basis for the formulations in this thesis. For the MIP-formulations in this thesis we will use

the binary variable y_v to denote vertex $v \in V$, and x_e or x_{ij} to denote the edge $e = (i, j)$. The binary variables are 1 if the vertex or edge is part of a solution, and 0 if not. For those formulation which require a root, we use an artificial root for unrooted test cases. An artificial root v_0 has the following properties [9]:

1. v_0 is part of any feasible solution.
2. v_0 has a weight of zero, i.e. $p(0) = 0$.
3. v_0 has a zero-weighted edge to every $v \in V$, i.e. $c_{0v} = 0 \quad \forall v \in V$.
4. One of the edges incident to v_0 is part of the solution, i.e. $\sum_{e \in \delta(0)} x_e = 1$.

For the multiple rooted generalization of the PCSTP, let $V_T \subseteq V$ denote the set of all roots, and let one of the roots be denoted v_0.

Let $V^* = V \setminus \{v_0\}$. All formulations in this thesis will then be a variant of the following,

$$\begin{aligned}
\min \sum_{v \in V} p_v(1 - y_v) + \sum_{e \in E} c_e x_e & \\
\sum_{e \in \delta(0)} x_e = 1, y_0 = 1 & \quad \text{if artificially rooted} \\
y_v = 1 & \quad \forall v \in V_T \text{ if multiple roots} \\
\sum_{v \in V^*} y_v \leq k & \quad \text{if maximum vertices in solution is k} \\
x \text{ defines a tree} & \\
y_v, x_e \in \{0, 1\} & \quad \forall v \in V, e \in E.
\end{aligned}$$

The maximum number of vertices in the solution constraint may be relevant for problems as network constructions where there is a restriction on how many commodities one can create due to regulations or time consumption. For simplicity in the formulations to come we do not count v_0. For our stomach cancer case we have decided to mainly control the number of vertices in the solution by a scaling of the edge weights.

2.2 Former Studies

The PCSTP has received a lot of attention, both by itself and as a part of bigger problems. We here give a short introduction to parts of what has been done. The first two paragraphs describe formulations which we will go further into later in this thesis, while the rest are included as a reference of other successful solution methods for the PCSTP.

Dummy point formulation

In 2006 Haouari et al. [5] solved a generalized PCSTP, where the total prize of the nodes for some subsets $V_K \subseteq V$ had to at least equal some preset prize quota q_k.

They introduced a dummy point formulation for their generalization of the PCSTP, which will be described for the PCSTP in Chapter 3. Lagrangian decomposition was done for two subproblems, one was the result of their generalized version of the PCSTP and one was obtained from the PCSTP itself. The dual problem was solved by the volume algorithm. A simple heuristic was introduced, which relies on creating a reduced instance during the solution process which can be solved to find a primal solution.

In 2007 Haourai et al. [6] showed that the PCSTP could be further decomposed into one more subproblem with this formulation. They showed theoretical and computational results for the strength of the decompositions, which will be used as arguments for choices made in this thesis. They also tested several variants of subgradient algorithms for their version of the PCSTP, where two has been implemented in this thesis.

MTZ-based formulations

In 2010 Haouari et al. [7] solved a quota version of the problem, where the sum of prizes in the optimal solution must be over a set threshold, $\sum_{v \in V_S} p(v) \geq Q$. They gave three distinct MIP formulations, based on the Miller-Tucker-Zemlin formulation proposed for the TSP in 1960 [13]. These were solved using a general purpose MIP-solver. This formulation is loosely based on introducing a variable which counts the number of edges from the root to any other vertex in the solution, ensuring connectivity.

These formulations were tested during the fall of 2012 as part of a preparation project for this thesis, using a general purpose MIP-solver. We managed to solve small instances to optimality. Together with the minimum adjacency reduction test we have now been able to obtain optimal solutions to more of our test cases.

Cutting plane method

In 2004 Lucena and Resende [10] created a cutting plane algorithm for the PCSTP. They used the GWMP formulation of the objective function, and solved the problem on an undirected graph. For any $S \subseteq V$, let $E(S) \subseteq E$ be the set of edges with both endpoints in S. Let $y(S) = \sum_{s \in S} y_s$ and $x(E(S)) = \sum_{e \in E(S)} x_e$. Then the constraints of the problem can be formulated as,

$$x(E) = y(V) - 1, \tag{2.1a}$$

$$x(E(S)) \leq y(S \backslash \{s\}), \qquad \forall s \in S, \forall S \subseteq V, \tag{2.1b}$$

$$x_e, y_v \in \{0, 1\} \qquad \forall e \in E, v \in V \tag{2.1c}$$

Equation (2.1a) makes sure the optimal solution has one more vertex than the number of edges, while (2.1b) make sure it's cycle free. Equation (2.1b) says that for any subset $S \subseteq V$, containing $q + 1$ vertices in the optimal solution, there are at most q edges with both endpoints in S in the optimal solution. Together these constraints guarantees the solution to be a tree. The idea is to add the exponentially

increasing number of equations from (2.1b) iteratively to the solution. Some new benchmark instances were solved to optimality.

In 2009 Cunha et al. [2] further exploited the structure of this formulation by Lagrangian relaxation, giving a relax and cut algorithm. They claimed that their heuristic outperformed most other heuristics from literature.

Branch and Cut

In 2006 Ljubić et. al [9] presented a branch and cut algorithm for the PCSTP. They reformulated the problem to a directed arc weighted graph. r is a root of the problem. The subtour elimination constraints used here are

$$\sum_{i \notin S, j \in S} x_{ij} \geq y_k \qquad \forall k \in S, r \notin S \forall S \subset V$$

Basicly this constraint says that for any proper subset $S \subset V$ not containing the root, if for the optimal solution there is a vertex in it, there has to be at least one arc into the set. The number of constraints are exponentially increasing, and they are implemented in a branch and cut framework. They also use other techniques to strengthen the formulation, not presented here.

Their method solved all test instances from literature to optimality, and they proposed some new harder problems. They solve large instances with up to 1825 vertices and 214095 edges to optimality within 12 hours. This seems to be the best exact solution algorithm available.

Reduction Tests

Several preprocessing techniques are presented and commonly used in literature [2, 9, 10]. The aim is to remove vertices and edges that are guaranteed not to be in the optimal solution, or to merge elements which are guaranteed to either be in the optimal solution together or not at all. We have implemented the minimum adjacency test, which helped us obtain optimal solutions for some test cases using the MTZ-formulation on a general purpose MIP-solver.

The reduction tests described below has earlier been proposed for the closely related Steiner Tree Problem in Graphs (STP). The first four was proposed for the PCSTP by [10], and also used in [2, 9]. The fifth was proposed for the PCSTP by [9] and also used by [2]. The tests should be run iteratively until none of them change the input graph.

Shortest path test: If there exist a path from vertex v_i to vertex v_j, where the sum of the edge costs $d_{ij} < c_{ij}$, where c_{ij} is the cost of edge (i, j), then edge (i, j) can be eliminated.

Cardinality-one test: For a vertex v_i of edge cardinality one, if the cost of that edge c_{ij} is greater than the prize p_i of the vertex, then v_i and (i, j) can be removed.

Cardinality-two test: For a vertex v_i of edge cardinality two, if the cost of the two incident edges c_{ij} and c_{ik} is greater than the prize p_i of the vertex, i.e. $c_{ij} > p_i$

and $c_{ik} > p_i$, then edge (i, j) and (i, k) can be merged into an edge $e = (j, k)$ of cost $c_{jk} = c_{ij} + c_{ik} - p_i$, or if e already exist $c_e = \min(c_{jk}, c_{ij} + c_{ik} - p_i)$. The reason for this is that either none or both of the edges (i, j) and (j, k) are part of the solution.

Cardinality-k test: For a vertex v_i of edge cardinality $k \geq 3$, denote the set of vertices adjacent to v_i by $V_{adj}(v_i) = \{v_1, v_2, ..., v_k\}$. Let $MST(K)$ be the cost of a minimum spanning tree of the distance subgraph of $K \subset V$. If

$$MST(K) \leq \sum_{w \in K} c_{iw}, \quad \forall K \subseteq V_{adj}(v_i), |K| \geq 3,$$

then, as in the cardinality-two test, v_i must have a cardinality of zero or two in an optimal solution. Each edge (i, j) and (i, k), where $j, k \in K$, can be replaced by an edge $e = (j, k)$ with cost $c_{jk} = c_{ij} + c_{ik} - p_i$, or if this edge already exist $c_e = \min(c_{jk}, c_{ij} + c_{ik} - p_i)$.

Minimum adjacency test: If adjacent vertices v_i and v_j exist such that

$$\min(p_i, p_j) > c_{ij} \text{ and } c_{ij} = \min_{ip \in E} c_{ip},$$

then v_i and v_j may be merged into one vertex of weight $p_i + p_j - c_{ij}$. If a vertex v_k was adjacent to both v_i and v_j, the edge of greatest weight is removed.

Net weight gain cardinality two path test: For any three vertices v_i, v_j and v_k where $(i, j), (i, k)$ and (j, k) exist. If

$$c_{ik} + c_{jk} - p_k < c_{ij} \text{ and } c_{ij} \geq \max(c_{ik}, c_{jk}),$$

then edge (i, j) can be removed.

The time complexity is $\mathcal{O}(|E|^2|V| + |E||V|^2 \log |V|)$ and is dominated by the shortest path test, but usually it runs much faster [9]. Worst case run time happens when the entire graph is reduced to one vertex, solving the problem.

In [14] they pose that these tests are not as advanced as their counterparts for the STP. They show a generalization which yield stronger tests. These tests are NP-hard, and solved using heuristics. Greater reductions than what has been obtained in [9] is showed for most instances. Some instances are even solved to optimality just using the reduction tests.

Chapter 3

Dummy Point Formulation for the PCSTP

In this Chapter we show an alternative formulation of the PCSTP, based on the work of Haouari et al. in [5, 6]. This formulation clearly shows some subproblems that the PCSTP consists of, making Lagrangian relaxation and decomposition possible.

3.1 Original formulation

It is obvious from the definitions of the PCSTP that if we know which vertices are part of the optimal solution, say V_S, the edges can be found by calculating the minimal spanning tree of the subgraph $G_S = (V_S, E_S) \subseteq G$. The following formulation exploits the MST structure of the problem.

Given a graph $G = (V, E)$. Let 0 denote the root, or if no root exist an artificial root. Then we let $G^* = (V^*, E^*)$ denote the subgraph of G where the root and its adjacent edges are removed.

In this formulation we add a dummy point d to V, giving $V^d = V \cup \{d\}$. This dummy point is restricted to be connected to the root, with an edge of weight zero. The dummy point also has an edge to all vertices in V^* with weight equal to the weight of the node, i.e. $c_{dj} = p_j \quad \forall j \in V^*$. Let E^d denote the union of E and the dummy point edges $(d, i) \quad \forall i \in V^*$. By restricting that any vertex in V^* connected to the dummy point, can not be connected to another vertex in V, and that the edges define a spanning tree, we get the following

$$\min \sum_{e \in E^d} c_e x_e \tag{3.1a}$$

$$\sum_{j \in V^*} x_{0j} = 1 \qquad \text{if artificially rooted} \tag{3.1b}$$

$$x_{0d} = 1 \tag{3.1c}$$

$$x_{dj} + x_{ij} \leq 1 \qquad \forall j \in V^*, (i,j) \in E \tag{3.1d}$$

$$x \qquad \text{defines a spanning tree on } G^d = (V^d, E^d) \tag{3.1e}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in E^d \tag{3.1f}$$

By comparing this formulation to the PCSTP-formulation, we see that if a dummy point edge x_{dv} is in the solution it compares to a vertex not being part of the PCSTP-solution. If $x_{dv} = 1$ a cost equal to p_v is added to the objective function, in the same way a vertex prize/penalty would be in the GWMP objective function if the vertex was not part of the solution. Thus this give the same objective value as the GWMP formulation. Figure 3.1 may be illustrative in understanding the concept.

Equation (3.1a), (3.1e) and (3.1f) together define a minimal spanning tree problem, complicated by the other constraints. As an MST is a relatively easy problem, this leads us to the use of Lagrangian relaxation. Constraint (3.1c) can easily be handled within the subproblem by initializing with it, so only (3.1b) and (3.1d) will be relaxed.

In [6] they show that this formulation can be further decomposed into a *Maximum-Weight Stable Set Problem* (MWSP) on a bipartite graph, from equation (3.1a),(3.1d) and (3.1f). Bipartite graphs are known to have totally unimodular incidence matrices, and decomposition will thus give no further gain in optimal value as opposed to relaxation. Therefore we relax (3.1d), and do not use Lagrangian decomposition.

Throughout the rest of this chapter, when referring to a number of vertices in the solution, we mean in the PCSTP-sense of Chapter 2 as illustrated in Figure 3.1.

3.2 Multiple root formulation

Some test instances may contain multiple roots. Let $T = \{v_{r1}, v_{r2}, ..., v_{rp}\}$ denote the set of all p root vertices except one which is denoted as main root v_{r0}. To induce a vertex $v \in T$ as part of the PCSTP-solution, it is sufficient to ensure that v is not connected to the dummy node, i.e. $x_{dv} = 0$. For Figure 3.1 we see that if we require $x_{d5} = 0$, then vertex 5 must connect to the PCSTP-solution, either directly or it may cause vertex 4 to also become part of the PCSTP-solution.

Ensuring that $x_{dv} = 0$ can be done by removing x_{dv} from the input graph G^d, and is thus easily handled. To induce multiple root vertices we thereby remove edges $(d, j) \quad \forall j \in T$ from E^d.

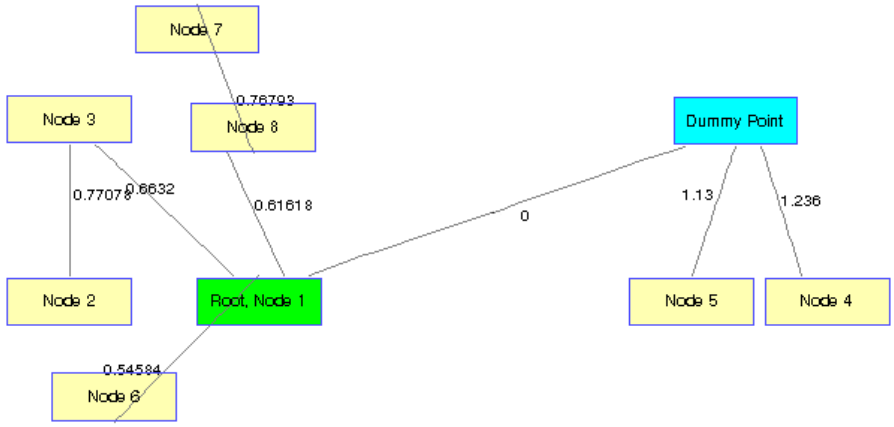

Figure 3.1: Illustration of a solution of the dummy point formulation. The dummy point is blue, and the root is green. The PCSTP-solution is found by removing the edge between the root and dummy point, and keeping only the part connected to the root vertex. Thus we here have six vertices in the solution, namely $1, 2, 3, 6, 7$ and 8.

3.3 Maximum number of vertices

We wish to see how we can include the maximum number of nodes constraint, normally written as

$$\sum_{j \in V^*} y_j \leq m \qquad \text{if maximum } m \text{ nodes of } n.$$

For the dummy point formulation we know that connecting a vertex to the dummy point is equivalent to saying that the vertex is not part of the solution. Thus requiring a maximum number of vertices in the solution, is equivalent to requiring a minimum number of vertices connected to the dummy point, i.e.

$$\sum_{j \in V^*} x_{dj} \geq n - m \qquad \text{if maximum } m \text{ nodes of } n. \tag{3.2}$$

3.4 Lagrangian Relaxation

We perform Lagrangian relaxation of (3.1d), (3.1b) and (3.2), and denote the Lagrangian multiplier with $u_{ij}^j \geq 0$ for (3.1d), v for (3.1b) and for (3.2) we use w. To be clear, u_{ij}^j refer to the constraint that vertex $j \in V^*$ connected to the dummy point d, should not include the edge (i, j).

As (3.1b) is an equality constraint, v is not restricted to be non-negative. Also constraint (3.2) is of form $Ax \geq b$ so we let $w \leq 0$. Let $q = n - m$, where n is the total number of elements in V.

We start by rewriting the objective function to clarify the subproblem,

$$\begin{aligned}
&\min \sum_{(i,j)\in E^d} c_{ij}x_{ij} + \sum_{j\in V^*} \sum_{(i,j)\in E} u_{ij}^j(x_{dj} + x_{ij} - 1) + f(v,x) + g(w,x) \\
=&\min \sum_{(i,j)\in E^d} c'_{ij}x_{ij} - \sum_{j\in V^*} \sum_{(i,j)\in E} (u_{ij}^j) - f^v - qg^w \\
=&\min \sum_{(i,j)\in E^d} c'_{ij}x_{ij} - K(u,v,w)
\end{aligned}$$

Then we get the following Lagrangian relaxation,

$$= \min \sum_{(i,j)\in E^d} c'_{ij}x_{ij} - K(u,v,w) \tag{3.3a}$$

subject to

$$x_{0d} = 1 \tag{3.3b}$$

$$x \quad \text{defines a spanning tree} \tag{3.3c}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E' \tag{3.3d}$$

where

$$c'_{ij} = c_{ij} + u_{ij}^i + u_{ij}^j \qquad \forall (i,j) \in E, i \neq 0 \tag{3.4a}$$

$$c'_{0j} = c_{0j} + u_{0j}^j + f^v \qquad \forall j \in V^* \cap \delta(0) \tag{3.4b}$$

$$c'_{dj} = c_{dj} + \sum_{(i,j)\in E} u_{ij}^j + g^w \qquad \forall j \in V^*. \tag{3.4c}$$

and

$$f(v,x) = \begin{cases} v(\sum_{j\in V^*} x_{0j} - 1) & \text{if artificially rooted} \\ 0, & \text{otherwise} \end{cases}$$

$$f^v = \begin{cases} v, & \text{if artificially rooted} \\ 0, & \text{otherwise.} \end{cases}$$

$$g(w,x) = \begin{cases} w(\sum_{j\in V^*} x_{dj} - q) & \text{if maximum nodes constraint included} \\ 0, & \text{otherwise} \end{cases}$$

$$g^w = \begin{cases} w, & \text{if maximum nodes constraint included} \\ 0, & \text{otherwise.} \end{cases}$$

For any given Lagrangian multipliers, the relaxation (3.3a)-(3.3d) is obviously an MST with the small modification that it is initialized with the edge $(0, d)$ already in the solution.

In practice the subgradient algorithm will make it more expensive to include the edges which breach the relaxed constraints, until we hopefully end up at a solution close to the optimal solution of the original formulation. In Figure 3.2 we see that the edge weight from the artificial root to vertex 7, c'_{07}, has been raised from its initial value of zero to 1.2423 in the solution. Notice also that we get the same PCSTP-solution as in Figure 3.1.

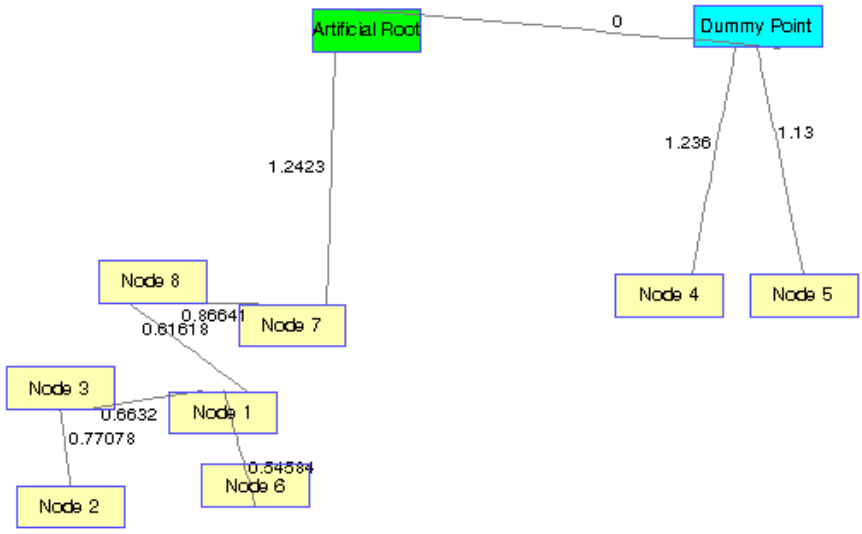

Figure 3.2: Illustration of a solution to the relaxed problem, where we get a feasible solution to the original problem, using an artificial root. The dummy point is blue, and the root is green. Notice that we here have the same edges and nodes in the solution as for the rooted instance of Figure 3.1. The edge weights are from the Lagrangian relaxation, and are therefore not equal to those of Figure 3.1.

3.5 Heuristics

When creating heuristic methods, numerous alternatives are often possible. We here present ideas for several tactics that make use of the primal information from the solution process. Two of the heuristics are implemented and results are given in Chapter 8, one which we have created and one from [5].

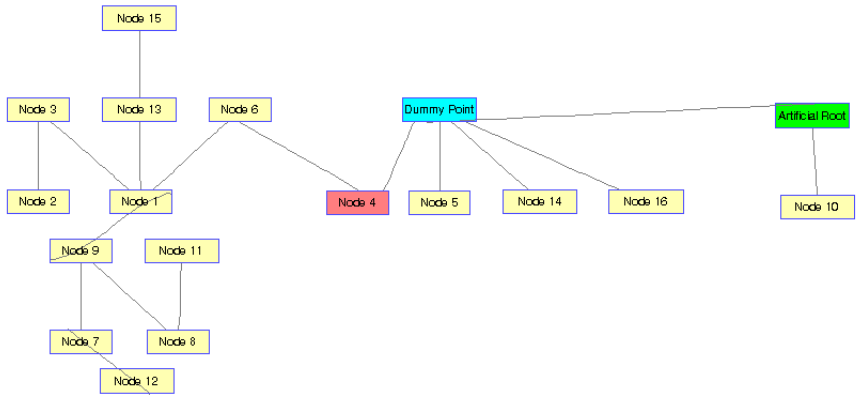

Figure 3.3: Illustration of a solution to the relaxed problem. Blue vertex is dummy point, green is the artificial root, and red is the vertex where we have a breached constraint. The breached constraints is: $x_{d4} + x_{4,6} \leq 1$. We see that the artificial root constraint is not breached here.

3.5.1 Heuristic based on shortest paths

Suppose we know which vertices are part of the solution beforehand. The PCSTP then reduces to finding the MST for the subgraph of G induced by these vertices.

The solution at any step of the dummy point formulation is an MST, so a substructure of the solution is likely to be close to a feasible solution. From Figure 3.3 a forest can easily be created by removing the dummy point and the artificial root, all their incident edges, and all vertices where the dummy point constraint (3.1d) is not breached. Then any cluster in this forest can be thought of as a PCSTP-solution, and also any merging of these clusters give a solution. The question then become how to merge these clusters.

We have tested some strategies based on greedy choices of edges. One strategy was to add the minimal cost edge between clusters (only applicable for complete graphs). Another was to add the cheapest edge connected to any of the clusters until they are merged. The positive here was that the structure was highly maintained so the run time was short, but both showed poor results.

By sacrificing some computational speed compared to the methods just mentioned, one can add clusters by the shortest path between them. This heuristic has four phases.

In the first phase before starting the subgradient algorithm, we calculate all pairs shortest paths, which can be done in $\mathcal{O}(n \log(n) + nm)$ time by Johnson's algorithm. This could have been done within the algorithm, as all pairs are not used, but worst case run time would likely be the same.

The rest of the heuristic is run each time we get a new intermediate solution in the subgradient algorithm. The second phase is a merging phase. Here we identify

all clusters, and add all vertices of one of them to a set P. Then we find the shortest path between the vertices of P and another cluster, and add all vertices in the shortest path and the cluster to P. This continues until no clusters remain. In the third phase the primal value is calculated by solving the MST on the subgraph of G induced by P. The last phase is a reduction phase, where we remove the leaves which have a higher cost than profit, as long as they are not roots.

Algorithm 2 Heuristic based on SPP

Before starting subgradient algorithm, calculate all pairs SPP.

Let x^k be an intermediate solution from the subgradient method. Let C be the set of all clusters obtained from x^k when removing the dummy point and the artificial root, all their incident edges, and all vertices where the dummy point constraint is not breached. Move all vertices of one of the clusters of C into P.

while C is not empty **do**

1: Find the shortest path between a cluster c and P.

2: Add all vertices in c and all vertices on the shortest path to P.

3: Delete c from C.

end while

3: Calculate primal value by solving the MST on the subgraph of G induced by P.

4: Remove leaves of the MST if the cost is greater than its profit, and the leaf is not a root.

A stronger heuristic might be possible by expressing the merging of clusters as a PCSTP. This could be done by reducing each cluster to one vertex, with weight equal to the sum of weights of its vertices minus the weights of its edges. The costs between these vertices could then be set equal to the shortest path between each cluster, possibly with the profits of the vertices along each shortest path added. Our heuristic finds the optimal solution of most of our test instances, and we have therefore not extended our implementation to do this.

3.5.2 Reduction based heuristic

A heuristic is proposed in [5], based on the observation that the edges of the optimal solution often will be present some time during the solution process. The heuristic saves up the edges used during the solution process, together with the edges of an MST to ensure connectivity, and solves this reduced instance. We use an Xpress Mosel implementation of the MTZ-formulation from Chapter 5 to solve this problem. This method requires only a small amount of extra time during the subgradient algorithm, but another solution method or heuristic is needed to solve the reduced instance after.

Chapter 4

Flow formulations

Using a flow-based formulation is a common way to ensure connectivity of a solution in optimization theory. An intuitive way of understanding a flow formulation is to think of a commodity being sent from a factory to a customer. If we create new variables that in some way require some artificial commodity being sent, or to *flow*, between all included vertices in the solution, this ensures connectivity.

Flow formulations require a directed graph, so we introduce a modified directed graph where each edge is replaced with two arcs in opposite directions. Let A(e) denote the set of the two arcs for each edge e. For the following formulations, V denote the set of all vertices and V^* is the set of all vertices except the root, possibly an artificial root. E denotes the set of edges.

We here present two flow based formulations for the PCSTP, and show possible Lagrangian relaxations. The second relaxation has been implemented and results are given in Chapter 8.

4.1 Formulation 1 - all to all

This formulation is based on a multicommodity flow formulation with a separate commodity for each pair of nodes. Here we have a separate flow from each vertex $u \in V$ to each vertex $w \in V, w > u$. As stated below it solves the unrooted PCSTP.

Original Formulation

$$\min \sum_{v \in V} p_v(1 - y_v) + \sum_{e \in E} c_e x_e \tag{4.1a}$$

$$f_a^{u,w} \leq x_e \qquad \forall u, w \in V, u < w, e \in E, a \in A(e) \tag{4.1b}$$

$$y_u + y_w - 1 \leq \sum_{a \in \delta^+(u)} f_a^{u,w} - \sum_{a \in \delta^-(u)} f_a^{u,w} \qquad \forall u, w \in V, u < w \tag{4.1c}$$

$$\sum_{a \in \delta^+(v)} f_a^{u,w} = \sum_{a \in \delta^-(v)} f_a^{u,w} \qquad \forall u, w, v \in V, v \neq u, v \neq w \tag{4.1d}$$

$$\sum_{v \in V} y_v = \sum_{e \in E} x_e + 1 \tag{4.1e}$$

$$0 \leq f_a^{u,w} \leq 1 \tag{4.1f}$$

$$x_e \in \{0, 1\}, y_v \in \{0, 1\} \tag{4.1g}$$

For any vertex u to be included in the solution, equation (4.1c) require that u behave as source for all flows $f_a^{u,w}$, and (4.1d) balances the flow through all other vertices except w which then behave as sink. Further (4.1b) makes sure there is no flow if an edge is not included in the solution. Thus including vertices require flow, and flow require edges. Equation (4.1e) ensures that the solution is a tree. This is not strictly necessary for the PCSTP, but our results have shown this to give a much stronger relaxation.

Lagrangian relaxation

By relaxation of (4.1b) and (4.1e), this formulation yield a set of decoupled *shortest path problems* (SPP). Let $\lambda_a^{u,w}$ denote the Lagrangian multiplier for (4.1b), and β for (4.1e). We start by rewriting the objective function,

$$\min \sum_{v\in V} p_v(1-y_v)+\sum_{e\in E} c_e x_e + \sum_{u\in V}\sum_{w\in V,u<w}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^{u,w}(f_a^{u,w}-x_e)$$
$$+\beta(\sum_{v\in V} y_v - \sum_{e\in E} x_e - 1)$$

$$= \min \sum_{v\in V} p_v + \sum_{v\in V}(\sum_{w\in V,u<w}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^{u,w} f_a^{u,w} + \beta y_v - p_v y_v)$$
$$+\sum_{e\in E}(c_e - \sum_{u\in V}\sum_{w\in V,u<w}\sum_{a\in A(e)} \lambda_a^{u,w} - \beta)x_e - \beta$$

$$= \min \sum_{v\in V}\sum_{w\in V,u<w}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^{u,w} f_a^{u,w} + K(y,x,\lambda,\beta)$$

The Lagrangian relaxation then have the following formulation,

$$= \min \sum_{v\in V}\sum_{w\in V,u<w}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^{u,w} f_a^{u,w} + K(y,x,\lambda,\beta) \tag{4.2a}$$

subjected to

$$y_u + y_w - 1 \le \sum_{a\in\delta^+(u)} f_a^{u,w} - \sum_{a\in\delta^-(u)} f_a^{u,w} \qquad \forall u,w\in V, u<w \tag{4.2b}$$

$$\sum_{a\in\delta^+(v)} f_a^{u,w} = \sum_{a\in\delta^-(v)} f_a^{u,w} \qquad \forall u,w,v\in V, v\neq u, v\neq w \tag{4.2c}$$

$$0\le f_a^{u,w}\le 1, \quad 0\le \lambda_a^{u,w}, \quad \beta \text{ free}, \tag{4.2d}$$

$$x_e\in\{0,1\}, y_v\in\{0,1\} \tag{4.2e}$$

Subproblem

We see that whenever $y_u = y_w = 1$, then (4.2) correspond to an SPP for vertex u and w. Thus the subproblem corresponds to solving SPPs for all pairs of vertices, and including vertices and edges if they pay off, i.e. $y_v = 1$ if

$$\sum_{w\in V,u<w}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^{u,w} f_a^{u,w} + \beta \le p_v$$

and $x_e = 1$ if

$$c_e \le \sum_{u\in V}\sum_{w\in V,u<w}\sum_{a\in A(e)} \lambda_a^{u,w} + \beta.$$

For solving this subproblem we need to solve $\mathcal{O}(n^2)$ one to one SPPs. As the edges are non negative, Dijkstra's algorithm can be used, giving a complexity of $\mathcal{O}(n^2 m \log(n))$ for each iteration of the subgradient algorithm.

4.2 Formulation 2 - root to all

For a rooted instance, or by introduction of an artificial root, the number of SPPs in the relaxation can be reduced. Let $0 \in V$ denote the root node, and $V^* = V \setminus \{0\}$ denote the set of all vertices except the root. In this formulation we have a separate flow from the root node to all other vertices. A generalized rooted version of the PCSTP is solved in [8] in a similar manner. As the root node is defined to be included in the solution, we use $\sum_{v \in V^*} y_v = \sum_{e \in E} x_e$ to ensure the solution is a tree.

Original Formulation

$$\min \sum_{v \in V^*} p_v(1 - y_v) + \sum_{e \in E} c_e x_e \tag{4.3a}$$

$$\sum_{e \in \delta(0)} x_e = 1 \qquad \text{if artificially rooted,} \tag{4.3b}$$

$$f_a^u \leq x_e \qquad \forall u \in V^*, e \in E, a \in A(e) \tag{4.3c}$$

$$y_u \leq \sum_{a \in \delta^-(u)} f_a^u - \sum_{a \in \delta^+(u)} f_a^u \qquad \forall u \in V^* \tag{4.3d}$$

$$\sum_{a \in \delta^+(v)} f_a^u = \sum_{a \in \delta^-(v)} f_a^u \qquad \forall u \in V^*, v \neq u \tag{4.3e}$$

$$\sum_{v \in V^*} y_v = \sum_{e \in E} x_e \tag{4.3f}$$

$$0 \leq f_a^u \leq 1 \qquad \forall u \in V^*, e \in E, a \in A(e) \tag{4.3g}$$

$$x_e \in \{0, 1\}, y_v \in \{0, 1\} \qquad \forall e \in E, v \in V \tag{4.3h}$$

Now constraint (4.3d) and (4.3e) require that there is a flow from the root to any other vertex in the solution, and (4.3c) require the edges to be included whenever there is flow. Constraint (4.3f) require the solution to be a tree and is not strictly necessary, but is included because it gives a stronger relaxation.

Lagrangian relaxation

We perform Lagrangian relaxation of constraints (4.3b) with Lagrangian multiplier γ, constraint (4.3c) with λ_a^u, and constraint (4.3f) with β. We first rewrite the

objective function,

$$\min \sum_{v\in V^*} p_v(1-y_v) + \sum_{e\in E} c_e x_e + \sum_{u\in V^*}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^u (f_a^u - x_e)$$
$$+\beta(\sum_{v\in V^*} y_v - \sum_{e\in E} x_e) + k^\gamma$$

$$= \min \sum_{v\in V^*} p_v + \sum_{u\in V^*}(\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^u f_a^u + \beta y_u - p_u y_u)$$
$$+\sum_{e\in E}(c_e - \sum_{u\in V^*}\sum_{a\in A(e)} \lambda_a^u - h_e^\gamma - \beta)x_e + l^\gamma$$
$$= \min \sum_{u\in V^*}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^u f_a^u + K_2(y,x,\lambda,\beta)$$

The Lagrangian relaxation is then given as

$$= \min \sum_{u\in V^*}\sum_{e\in E}\sum_{a\in A(e)} \lambda_a^u f_a^u + K_2(y,x,\lambda,\beta) \tag{4.4a}$$

subject to

$$y_u \le \sum_{a\in\delta^-(u)} f_a^u - \sum_{a\in\delta^+(u)} f_a^u \qquad \forall u \in V^* \tag{4.4b}$$

$$\sum_{a\in\delta^+(v)} f_a^u = \sum_{a\in\delta^-(v)} f_a^u \qquad \forall u,v \in V^*, v \neq u \tag{4.4c}$$

$$0 \le f_a^u \le 1 \tag{4.4d}$$

$$0 \le \lambda_a^u \tag{4.4e}$$

$$x_e \in \{0,1\}, y_v \in \{0,1\} \tag{4.4f}$$

where

$$k^\gamma = \begin{cases} \gamma(1 - \sum_{e\in\delta(0)} x_e), & \text{if artificially rooted} \\ 0, & \text{otherwise} \end{cases}$$

$$l^\gamma = \begin{cases} \gamma, & \text{if artificially rooted} \\ 0, & \text{otherwise} \end{cases}$$

$$h_e^\gamma = \begin{cases} \gamma, & \text{if artificially rooted and } e \in \delta(0) \\ 0, & \text{otherwise.} \end{cases}$$

Subproblem

We see that if $y_v = 1$ in equation (4.4b), then for a given set of Lagrangian multipliers formulation (4.4) corresponds to a shortest path problem for each $u \in V^*$,

with edge costs of λ_a^u. After solving the SPP for each u, a node or edge is included in the solution if it pays off, i.e. $y_v = 1$ if

$$\sum_{e \in E} \sum_{a \in A(e)} \lambda_a^v f_a^v + \beta \leq p_v$$

and $x_e = 1$ if

$$c_e \leq \sum_{u \in V} \sum_{a \in A(e)} \lambda_a^u + h_e^\gamma + \beta.$$

The edges are non negative so Dijkstra's algorithm can be used, giving a complexity of $\mathcal{O}(nm \log(n))$ for each iteration of the subgradient algorithm.

A simple heuristic of creating an MST out of the vertices included in the subproblem can be done in an additional $\mathcal{O}(m \log(n))$ time.

Chapter 5

MTZ-formulation

The Miller-Tucker-Zemlin (MTZ) formulation was used for the PCSTP in [7]. It ensures that the solution is a tree, by requiring that a set of bounded variables $u_v \quad \forall v \in V^*$ increase when we move away from the root node in the solution.

This is a rooted formulation, so for a unrooted instance we use an artificial root. We create a bidirected graph from G, called $G_A = (V, A)$. Denote by $\delta^+(j)$ the set of all arcs out of vertex j, and $\delta^-(j)$ the set of arcs into j. Let the cost of an arc c_a be equal to the cost of the corresponding edge c_e. We let $x_a = x_{ij}$ denote an arc from i to j. Then the MTZ conditions can be stated as,

$$\min \sum_{v \in V^*} p_v(1 - y_v) + \sum_{a \in A} c_a x_a \tag{5.1a}$$

$$\sum_{a \in \delta^+(0)} x_a = 1 \qquad \text{if artificially rooted} \tag{5.1b}$$

$$\sum_{a \in \delta^-(j)} x_a = y_j, \qquad \forall j \in V^* \tag{5.1c}$$

$$x_{ij} + x_{ji} \leq y_j, \qquad \forall j \in V \tag{5.1d}$$

$$u_i - u_j + y_j \leq n(y_i - x_{ij}), \qquad \forall (i,j) \in A \tag{5.1e}$$

$$y_j \leq u_j \leq n y_j, \qquad \forall j \in V^* \tag{5.1f}$$

$$u_0 = 0, y_0 = 1 \tag{5.1g}$$

$$x_{ij}, y_j \in \{0, 1\}, \qquad \forall (i,j) \in A, \forall j \in V \tag{5.1h}$$

From (5.1e) we notice that when $x_{ij} = 1$, then $y_i = y_j = 1$ according to equation (5.1d), and

$$u_j \geq u_i + 1, \quad \forall (i,j) \in A$$

Thus whenever an arc x_{ij} is part of the solution, the MTZ variables increase by at least one from i to j. We also see that for (5.1e) $x_{ij} = 0$ give

$$u_i - u_j + y_j \leq n y_i$$

which obviously hold for every combination of $y_i, y_j \in \{0, 1\}$, remembering that equation (5.1f) requires $u_j = 0$ whenever $y_j = 0$.

Because it is also bounded from below and above (5.1f), this guarantees a cycle free solution. Further as (5.1c) require that all vertices except the root has an arc into it, the solution is connected and a tree.

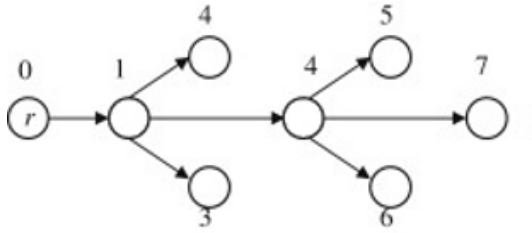

Figure 5.1: Example of an MTZ solution.

This formulation was implemented in FICO Xpress Mosel in the fall of 2012 as part of a preparation study for this thesis. In the results chapter we compare the LP-relaxation of the MTZ-formulation with the Lagrangian relaxations presented in earlier chapters.

Chapter 6

Test Instances

In this chapter we give an introduction to the test cases which are used in the remainder of this thesis. Any reference to number of vertices or edges exclude the artificial root.

6.1 Stomach Cancer

The most common stomach cancer, gastric adenocarcinoma, is histologically (on a microscopic level) divided into two major types, intestinal and diffuse. Histologically there are significant differences between intestinal and diffuse cancer, which have also been seen at a molecular level, proposing that there also is a molecular difference between intestinal and diffuse cancer. However, Vidar Beisvåg is working on a hypothesis that this molecular difference may not be related to the histological classification by itself, but may be caused by the present of so called signet ring cells. Only a subgroup of the diffuse cancers contain signet ring cells, and when intestinal and diffuse cancers are compared at a whole genome level, no or just a little difference is found. But, when diffuse cancers with and without signet ring cells are compared a large difference is shown. Which may indicate that the molecular differences previously shown between diffuse and intestinal cancer, may actually come as a result of those diffuse cancers containing signet ring cells.

We will here define a PCSTP for this case. The goal is to find a subgroup of genes, which may indicate an important molecular difference between diffuse cancer with and without signet ring cells. A significant amount of signet ring cells are generally associated with a worse prognosis for a patient, and therefore finding these genes could potentially lead to a better treatment.

Our data come from diffuse tumors of 29 operated individuals, 10 with signet ring cells and 19 without. For the 29 tumors we have the relative amount of mRNA molecules in a sample for 16110 genes, giving a matrix of size 29×16110. All of the following calculations come from this data set.

Let d_{gi} be the data for individual i and gene g. Let S denote the set of indi-

viduals with signet ring cells, and D the set without. We let a gene g correspond to a vertex v in the PCSTP. It's weight p_v is determined by the so called log fold change between the two cancer types, which here is the difference of the logarithm of the average value between the groups. Only genes with a higher average in S than in D is included, giving $p_v \geq 0$.

$$p_v = \log_2(\frac{\sum_{i \in S} d_{gi}}{|S|}) - \log_2(\frac{\sum_{i \in D} d_{gi}}{|D|}).$$

Thus a high difference in expression level for a gene between the two groups will give a high weight, making it more likely to be part of a PCSTP solution. A normal starting point for a biologist is to start at the top of the log fold change list to find interesting genes for further exploration.

Biological processes are normally dependent on several genes, and we want to reward clusters of genes which are dependent of each others. By using correlation one can identify clusters of coregulated genes. Thus for the edges, let k_{ij} be the Pearson correlation factor between gene i and j obtained from the data set, and let SF be a positive scaling factor. We then let c_{ij} be the weight between vertex i and j, and calculate the edge weights as,

$$c_{ij} = (1 - k_{ij}^2)SF$$

By doing this a big positive or negative correlation between two genes correspond to a low cost between two vertices in the PCSTP, making it more likely that these two genes will be part of a solution simultaneously.

By using correlation one can identify clusters of genes, and looking at the log fold change gives genes which are expressed differently in the two groups. Through this modeling we hope to do both, finding clusters of coregulated genes which are expressed differently for the two groups.

Using the log fold change has some weaknesses from a biological point of view. When a tumor is analyzed, only a slice of the tumor is used and this slice might not have a representative amount of signet ring cells. Thus the fold change may be misleading, as it may give an undeserving high or low weight. The correlation between genes will however not be affected by this, and will smooth out some of this effect.

More weaknesses exist, but is out of scope for this thesis. We do however note that solving this problem as a PCSTP, as opposed to using standard statistical analysis, give a high degree of flexibility. Vertex and edge weights can easily be adjusted if one wants other relationships between genes to be taken into account. Further it is easy to impose restrictions on a solution within the optimization framework, which will in part be done later for the dummy point formulation.

The full data set consist of 16110 genes. This gives a complete graph of 16110 vertices and almost 130 million edges. A trial run have been done on the full set, giving memory problems and a huge time consumption.

A subset of 187 genes have been picked by Arnar Flatberg. This is a complete graph, giving 187 vertices and 17391 edges. From this set we also created smaller

test instances from the first 16, 32, 64 and 128 vertices, with respectively 120, 496, 2016, 8128 edges. In the results chapter we denote these sets with a prefix S16, S32 etc, and a suffix SF20 or SF200 based on what scaling factor has been used. The test case S16SF20 then consist of 16 vertices and a scaling of 20 on the edge weights.

Qualified guesses

For this test instance there is a possibility that the user has a qualified guess of some genes which are relevant, or the user might wish to see which other genes seem to be important given a set of genes specified by the user. This can be done by the generalized multiple roots PCSTP.

Maximum vertices constraint

As we mainly want a small number, $m < n$, of candidate genes, we also test with a constraint for the number of vertices included in the solution

$$\sum_{j \in V^*} y_j \leq m$$

Notice that if the scaling factor on the edges is too small, the optimal solution will only contain the vertices of highest weight, so a sufficiently big scaling factor is still necessary. Due to this weakness, we have mainly focused on controlling the number of genes in the solution by adjusting the scaling factor. Further we have not created a heuristic which incorporates this constraint. Some dual results are shown in Chapter 8, mainly to show the strength of the formulations when using the flexibility within optimization.

6.2 Literature instances

We test our formulations against benchmark instances also solved in [9, 10]. Class P and K are among the easier instances solved in these articles. These instances can be found at *http://homepage.univie.ac.at/ivana.ljubic/research/pcstp/*. Notice that these graphs are sparse compared to the cancer research case.

	Vertices	Edges
P100.1	100	284
P200	200	587
P400	400	1200
K400	400	1515

Table 6.1: Test instances from literature, with number of vertices and edges.

Chapter 7

Implementation

In this chapter we show how the different formulations and relaxations have been implemented. In Section 7.1 we show the implementation of the dummy point relaxation, in Section 7.2 the relaxation of the second flow formulation, and in Section 7.3 the implementation in Mosel of the MTZ formulation.

The Lagrangian relaxations are solved by the subgradient method, as outlined in Section 1.1.4. They are implemented in Matlab R2012a, and built-in algorithms for solving the MST and SPP are used, namely GraphMinSpanTree and GraphShortestPath. The minimum adjacency reduction test is also implemented in Matlab. Details about this implementation is skipped. For these tests we use a Dell Optiplex 980, with quad core processor Intel Core i7 CPU 860 @ 2.80 GHz x 4, and 16GB of RAM, with Ubuntu 12.04 64-bit operative system.

Flow formulation 1 and 2, and the MTZ-conditions are implemented in FICO Xpress Mosel 3.4.1 on a Dell Optiplex 755 with Windows 7 SP1, dual core (2.67GHz) and 4GB of RAM.

Below we use the notation `(i,j)` for row i and column j of a 2d-array, and (i, j) for an edge from vertex i to j. A colon in a 2d-array `b(:,i)` means the i-th column, and two colons in a 3d-array `c(:,:,i)` means the i-th 2d-array. Let n and m be the number of vertices and edges in the input graph G.

7.1 Dummy Point Formulation

In Algorithm 3 we give the subgradient method for the dummy point formulation, and we will thereafter explain how we build $\texttt{Ax}_\texttt{k}$. Together with Chapter 3 we think these explanations are sufficient for a basic understanding of our implementation.

For the SPP-heuristic we need to solve all pairs SPP. This has a time complexity of $\mathcal{O}(nm + n\log(n))$ using for instance Johnson's algorithm. Matlabs implementation of this algorithm does however not return the paths, only the costs. We therefore use Dijkstras algorithm with a complexity of $\mathcal{O}(nm\log(n))$.

To solve the MST subproblem we have a 2d-array `c` to store all the original edge

costs, and a 2d-array $\mathtt{G_k}$ to handle the updated costs during the solution process. GraphMinSpanTree takes $\mathtt{G_k}$ as input and outputs a sparse 2d-array $\mathtt{x_k}$ with n non zero elements. An element in row `(i,j)` of $\mathtt{x_k}$ corresponds to an edge (i, j) in the current solution, with value equal to the current cost of that edge. Prims algorithm is used, with run time $\mathcal{O}(m \log(n))$.

Algorithm 3 Subgradient method for dummy point formulation

Step 0: Initialize all matrices and variables.
for k=0 **to** Maximum iterations **do**
Step 1a: Update $\mathtt{G_k}$ according to the costs in Section 3.4.
Step 1b: Input $\mathtt{G_k}$ in GraphMinSpanTree to get $\mathtt{x_k}$.
Step 1c: Update current dual value $\mathtt{z_k}$ = sum($\mathtt{x_k}$) - sum($\mathtt{u_k}$) -$\mathtt{v_k}$ - $\mathtt{qw_k}$.
Step 1d: Update lower bound.
Step 1e: If the lower bound has not increased in a set number of iterations, reset matrices to earlier point and half the step size scalar l_k.
Step 2a: Calculate $\mathtt{Ax_k}$, $\mathtt{v_k}$ and $\mathtt{w_k}$.
Step 2b: Check for feasibility or run heuristic to get a primal value.
Step 2c: Update upper bound.
Step 3a: Calculate the subgradient $\mathtt{gu_k}$, $\mathtt{gv_k}$ and $\mathtt{gw_k}$.
Step 3b: Calculate the search direction $\mathtt{du_k}$, $\mathtt{dv_k}$ and $\mathtt{dw_k}$.
Step 3c: Calculate the step length $\mathtt{t_k}$.
Step 4: If the bounds are within an acceptable limit, then stop.
Step 5: Update Lagrangian multipliers $\mathtt{u_k}$, $\mathtt{v_k}$ and $\mathtt{w_k}$.
end for

Building $\mathtt{Ax_k}$

Each element of the relaxed dummy point constraint is handled by a 2d-array $\mathrm{Ax_k}$ as follows, where d denotes the dummy point,

$$\mathtt{Ax_k(i,j)} \leftarrow x_{di} + x_{ij}$$

The reason for handling Ax, which is actually a vector, as a 2d-array is to easier let the elements correspond to the edges. For sparse input graphs, some elements will thus be unused.

Matrix $\mathtt{Ax_k}$ is built from the current solution $\mathtt{x_k}$. An edge (i, j) from $\mathtt{x_k}$, where $i, j \neq dummy$, has two contributions to $\mathtt{Ax_k}$, namely for the following elements

$$\mathtt{Ax_k(i,j)} \leftarrow x_{di} + x_{ij}$$
$$\mathtt{Ax_k(j,i)} \leftarrow x_{dj} + x_{ij}$$

An edge (d, i) has contributions to all $(i, j) \in E$. Assuming an ordered list where the first edge is $(i, 1)$ and the last edge is (i, n), it contributes to all of the following

$$\begin{aligned} \texttt{Ax}_\texttt{k}\texttt{(i,1)} &\leftarrow x_{di} + x_{i1} \\ &\vdots \\ \texttt{Ax}_\texttt{k}\texttt{(i,n)} &\leftarrow x_{di} + x_{in} \end{aligned}$$

The i-th row of the incidence matrix `b` gives the indices to update for all (d, i). This results in Algorithm 4, where `xr` and `xc` are arrays with row and column indices of non zero elements of $\texttt{x}_\texttt{k}$, such that `xr(1)` and `xc(1)` give the first edge of $\texttt{x}_\texttt{k}$ in some order. Note that by construction the dummy point will always be in `xr`, so we only test if `xr(a)` $\neq$ *dummy*.

Algorithm 4 Building $\texttt{Ax}_\texttt{k}$

```
Axk = zeros(n,n);
for a = 1:length(xr) do
    if xr(a) ≠ dummy then
        Axk(xr(a),xc(a)) = Axk(xr(a),xc(a))+1
        Axk(xc(a),xr(a)) = Axk(xc(a),xr(a))+1
    else
        Axk(xc(a),:) = Axk(xc(a),:) + b(xc(a),:)
    end if
end for
Axk(root,:) = 0;
```

7.2 Flow formulation 2

Our implementation for Flow Formulation 2 requires a complete graph, and only the stomach cancer case is tested. As the results were poor, we have not used time to generalize the implementation for sparse graphs. We do not outline the entire solution process for this formulation, but Algorithm 5 shows how we solve the most interesting part of the subproblem.

For λ_a^u and f_a^u, where $a = (i, j)$, we use three dimensional arrays $\texttt{L}_\texttt{k}\texttt{(i,j,u)}$ and $\texttt{F}_\texttt{k}\texttt{(i,j,u)}$. GraphShortestPath takes as input $\texttt{L}_\texttt{k}\texttt{(:,:,u)}$ with a source (`root`) and sink (`u`) vertex. As $\texttt{L}_\texttt{k} \geq 0$ Dijkstra is used with a run time of $\mathcal{O}(m \log(n))$ for each u. This totals to a run time of $\mathcal{O}(nm \log(n))$ for each iteration. The subgradient method uses most of its time in GraphShortestPath.

In the algorithm below we show how $\texttt{F}_\texttt{k}\texttt{(i,j,u)}$ is built, and how vertices are chosen. Edges are chosen separately.

Algorithm 5 Building `F_k(i,j,u)` and choosing vertices

Let β be the Lagrangian multiplier for $\sum y - \sum x = 1$
for `u` $\in V \setminus \{root\}$ **do**
  [`COSTu PATHu`] = GraphShortestPath(`L_k(:,:,u)`,`root`,`u`).
  if `COSTu` + β <= p_u **then**
    $y_u = 1$.
    `F_k(i,j,u)` = 1 for all (i, j) along the path in `PATHu`.
  end if
end for

This can potentially be done in parallel to improve the run time, which with n cores would be equal to that of the dummy point.

7.3 MTZ and Flow Formulations in Mosel

The MTZ formulation was implemented in FICO Xpress Mosel as part of a specialization project for this thesis.

Truls Flatberg has implemented the first flow formulation in Mosel, and this has been altered by the author to apply for the second formulation. Mosel quickly struggle with memory problems when the problem size increases. The implementation has been used to verify that the Lagrange relaxation is equal to the LP-relaxation for all instances which we were able to test.

Chapter 8

Results

In this Chapter we give results for the formulations and relaxations. We compare primal and dual bounds, solution methods and time consumption. A new cost effective heuristic for the unrooted stomach cancer instances is proposed from these results.

We give total run time in seconds for most test cases. Parts of our implementations can be improved, so we also include run times used in GraphMinSpanTree and GraphShortestPath as these implementations are thought to be fairly effective.

As the dummy point and flow formulations seem to have duality gaps, the optimal solutions z_{opt} come from the Mosel implementation of the MTZ-formulation. In the tables we use italics on instances where we have not obtained an optimal solution.

In all the following results we use a maximum of 500 iterations, and halve the step size if there has been no improvement for 20 iterations. We start with step length scalar $l_0 = 2$.

8.1 Dummy Point Formulation

In this section we give results for the dummy point formulation. We show results for unrooted and multiple rooted instances, and by the use of the maximum number of nodes constraint.

If not otherwise stated, the initial heuristic to obtain a primal bound is based on the best bound obtained from an MST and the single node of highest weight.

8.1.1 Unrooted PCSTP

In Table 8.1 we compare different strategies for finding the dual value of the PCSTP. Table 8.2 shows dual and primal bounds by the use of the SPP-heuristic, and in Table 8.3 we give primal bounds from the Reduction based heuristic.

In Table 8.1 we see that SG1 finds a slightly tighter bound than SG2 for most instances. A comparison of convergence is shown in Figure 8.2 for S64SF20, and in Figure 8.3 for P.100.1 using heuristics. SG1 is also slightly faster than SG2, as the search direction is equal to the subgradient. We have included the run time used in GraphMinSpanTree for solving the subproblem. For the complete graphs we see that most of the run time is used in GraphMinSpanTree, which shows that our implementation of the subgradient method is fairly efficient. Looping gives a slightly tighter bound for almost all instances, but the added run time makes looping a poor alternative for solving the relaxations. There seems to be a duality gap for most instances, such that for example implementation into a branch-and-bound framework is necessary if one wants to guarantee optimality.

In Table 8.2 we see that our SPP-heuristic finds the optimal solution for all the stomach cancer instances in less than 5 iterations. In two out of the three literature instances that we have optimal values for, it also finds these within 500 iterations. Bounds for P.100.1 are shown in Figure 8.3. We see that the run time increases compared to when we used the optimal solution as upper bound. Only a part of the extra time is used to solve the SPPs, so the implementation of the heuristic can probably be improved.

Table 8.3 shows the results of the edge reduction heuristic from [5]. It finds the optimal solution for all but one of the stomach cancer instances, and performs better than the SPP-heuristic on some of the literature instances. This heuristic saves up all edges used during the solution process, and to ensure connectivity it merges them with the edges of an MST. As the number of edges from the MST is $n - 1$, we see that for the stomach cancer instances only a small amount of edges differ from the MST. This leads us to believe that for further work a strong heuristic could be made independently of the dummy point formulation, by simply finding the MST of the input graph, and solving the PCSTP over the edges from this MST.

This is also supported by the SPP-heuristic, as it finds the optimal solution in the first iteration for several of the instances. In the first iteration of the dummy point formulation, all Lagrangian multipliers are zero, and thus the cost from the artificial root to all other vertices is zero. As the edge weights of the input graph for the stomach cancer instances are in practice greater than zero, the solution in the first iteration $\mathtt{x}_0$ will then be as in Figure 8.1, where all vertices are connected to the artificial root. We then remove the artificial root and the dummy point, and merge the remaining vertices. For the stomach cancer instances this will in practice always be all of the vertices of the input graph. Thus a minimal spanning tree over all vertices, with removal of non profitable leaves, finds the optimal solution of seven out of the ten stomach cancer instances.

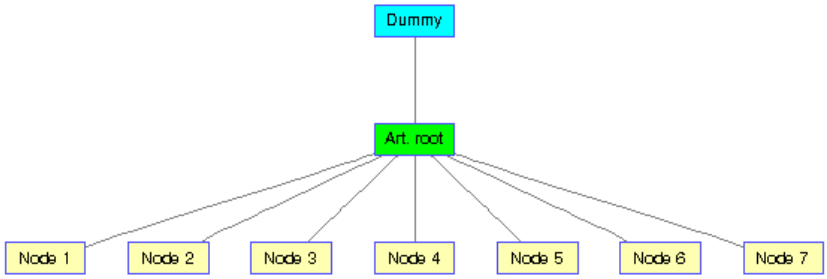

Figure 8.1: Illustration of the solution from the first iteration $\mathtt{x}_0$, for the artificially rooted dummy point formulation. Green is the artificial root, and blue is the dummy point.

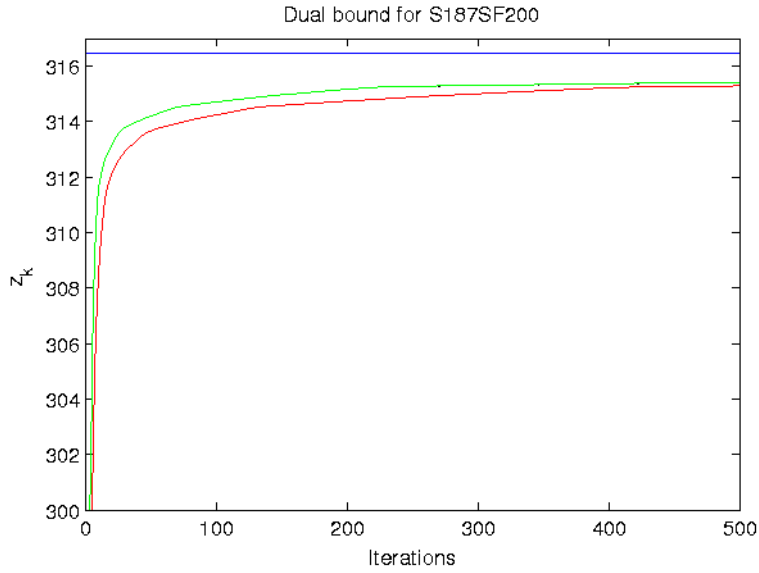

Figure 8.2: Dual bound for S187SF200, when using optimal solution as upper bound (in blue). In green we see the bound using SG1 and in red the bound from SG2. We have a fairly monotonic increase, as the dual values (in black) are mostly hidden behind the lower bounds. This is due to the good initial upper bound.

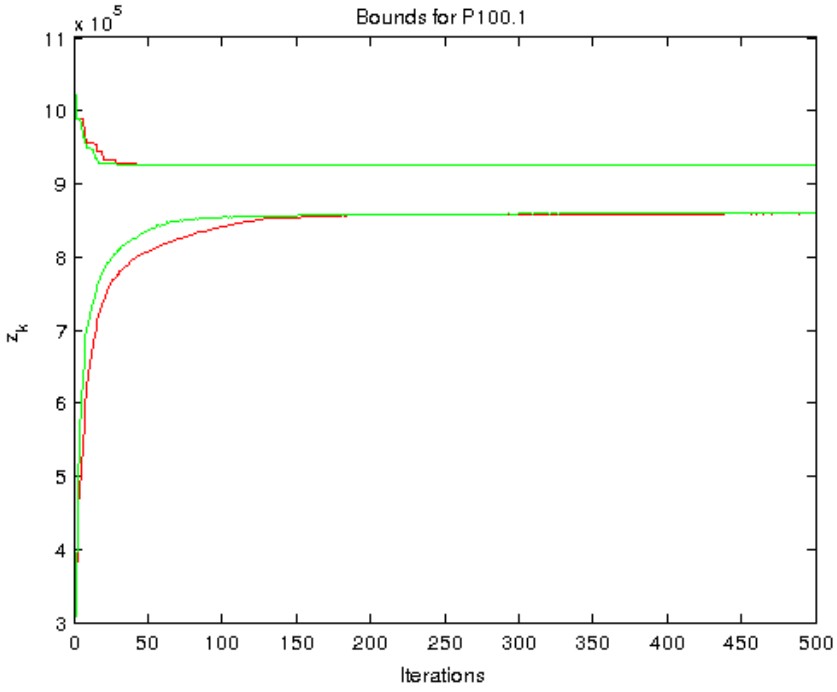

Figure 8.3: Bounds for P.100.1 using the SPP-heuristic. Upper and lower bounds when using SG1 is shown in green, and SG2 in red. As the heuristic finds the optimal solution in the 29th iteration, there seems to be a duality gap.

Dual values using an artificial root, UB = $\mathbf{z_{opt}}$

	SG1	SG2	z_{opt}	LB/z_{opt}	t_{SG1}	t_{SG2}	t_{MST}
S16SF20	17.9506	17.9154	18.1439	0.9893	0.4	0.4	0.1
S32SF20	31.2822	31.2231	32.0602	0.9757	0.5	0.6	0.2
S64SF20	58.4055	58.2511	59.2132	0.9864	1.1	1.2	0.5
S128SF20	103.4995	103.2695	104.6270	0.9892	2.6	2.7	2.3
S187SF20	140.4932	140.1984	141.8218	0.9906	4.9	5.1	4.8
S16SF200	22.8100	22.8100	22.8100	1	0.1	0.2	0.03
S32SF200	51.9394	51.9392	51.9394	1	0.2	0.3	0.1
S64SF200	108.1637	108.2387	108.7540	0.9953	1.1	1.1	0.6
S128SF200	212.8768	212.8096	213.5898	0.9967	2.6	2.7	2.3
S187SF200	315.4292	315.2925	316.4688	0.9967	5.0	5.1	4.9
P100.1	860216	859194	926238	0.9287	1.1	1.2	0.2
P200	1261298	1253423	1317874	0.9571	2.5	2.7	0.5

Dual values using looping, UB = $\mathbf{z_{opt}}$

	SG1	SG2	z_{opt}	LB/z_{opt}	t_{SG1}	t_{SG2}
S16SF20	17.9797	17.9373	18.1439	0.9910	0.9	1.0
S32SF20	31.2876	31.2253	32.0602	0.9759	11	11
S64SF20	58.4136	58.2559	59.2132	0.9865	58	58
S128SF20	103.6358	103.5844	104.6270	0.9905	333	370
S187SF20	140.8777	140.8462	141.8218	0.9933	1008	1077
S16SF200	22.8100	22.8100	22.8100	1	0.2	0.2
S32SF200	51.9394	51.9394	51.9394	1	0.2	0.2
S64SF200	108.2409	108.2409	108.7540	0.9953	3.2	3.0
S128SF200	212.9285	212.8479	213.5898	0.9969	13.5	13.6
S187SF200	315.4706	315.3660	316.4688	0.9968	40	40
P100.1	860145	859903	926238	0.9286	127	146
P200	1262828	1256964	1317874	0.9582	515	538

Table 8.1: Dual bounds using the dummy point formulation, where the optimal solution is used as upper bound. Above we use an artificial root, and below we loop through all vertices as root. For the LB/z_{opt}-column we use the best of the two values obtained from SG1 and SG2 as LB. For the columns t_{SG1} and t_{SG2} we give the total run time in seconds for each of the subgradient methods, and in the t_{MST}-column the time used in GraphMinSpanTree for the subproblem with SG1.

SPP-heuristic, artificial root

	UB	z_{opt}/UB	its	t	t_{SPP}
S16SF20	18.1439	1	1	0.6	0.02
S32SF20	32.0602	1	3	0.9	0.04
S64SF20	59.2955	1	5	1.7	0.1
S128SF20	104.6270	1	1	4.3	0.6
S187SF20	141.8218	1	1	9.0	1.6
S16SF200	22.8100	1	4	0.3	0.02
S32SF200	51.9394	1	1	0.5	0.04
S64SF200	108.7540	1	1	1.4	0.1
S128SF200	213.5898	1	1	3.2	0.6
S187SF200	316.4688	1	1	6.6	1.6
P100.1	926238	1	29	3.0	0.2
P200	1340946	0.9828		8.0	0.6
P400	2527372			21	4.2
K400	384605			22	4.3

Table 8.2: Primal bounds using the SPP-heuristic. Dual bounds are more of less equal to those of Table 8.1, and are not given. In the its-column we state how many iterations are needed before the heuristic finds the optimal solution. The t-column gives total run time, and t_{SPP} gives time for calculating the shortest paths.

Edge reduction-heuristic, artificial root

	Edges	UB	z_{opt}/UB	t
S16SF20	18	18.1439	1	0.1
S32SF20	36	32.0602	1	0.1
S64SF20	69	59.2955	0.9986	0.2
S128SF20	128	104.6270	1	0.4
S187SF20	187	141.8218	1	0.6
S16SF200	15	22.8100	1	0.1
S32SF200	31	51.9394	1	0.1
S64SF200	64	108.7540	1	0.2
S128SF200	128	213.5898	1	0.5
S187SF200	187	316.4688	1	0.6
P100.1	128	926238	1	44
P200	240	1317874	1	54
P400	484	2462863		600
K400	557	385775		600

Table 8.3: Results from the edge reduction-heuristic. Number of edges left is given in Edges. UB give the best integer bound found. Maximum time is set to 600 seconds. This only includes the time used in Xpress Mosel.

8.1.2 Multiple rooted PCSTP

To test the multiple rooted PCSTP, we choose to root the five vertices [1 2 3 4 5]. At least one of these vertices is not part of the known optimal solutions for the corresponding unrooted PCSTPs. SG1 is used for all instances

Results for the SPP-heuristic are given in Table 8.4. We notice that the SPP-heuristic finds close to optimal solutions, but does not perform as well as for the unrooted PCSTP. The optimal solution is found by feasibility in two instances and is thus guaranteed, and by the heuristic in seven of the instances. For four of the instances where we know the optimal solution, the heuristic does not find it.

In Table 8.5 results from the edge reduction heuristic are given. Also here we see tight bounds, which are mostly found rapidly. We see that for some instances the reduced graph only contain the edges of a minimal spanning tree, while for most instances some extra edges are also included. For K400 no integer solution was found in Xpress Mosel.

Bounds with multiple roots, and SPP-heuristic

	LB	UB	LB/UB	z_{opt}/UB	t
S16SF20	*1*		1	1	0.2
S32SF20	34.3297	*200*	0.9967	1	0.6
S64SF20	59.0131	*411*	0.9939	1	1.2
S128SF20	104.4092	*1*	0.9964	1	3.1
S187SF20	141.7066	*1*	0.9980	1	6.2
S16SF200	*59*		1	1	0.3
S32SF200	132.9860	*14*	0.9967	1	0.8
S64SF200	179.7599	189.6171	0.9480	0.9866	2.3
S128SF200	277.5506	293.6851	0.9451	0.9822	4.8
S187SF200	374.9121	390.0514	0.9611	0.9933	8.6
P100.1	1092882	*16*	0.9818	1	3.4
P200	1342438	*132*	0.9932	1	6.9
P400	2573965	2626860	0.9798	0.9932	21
K400	280531	502180	0.5586		23.5

Table 8.4: Bounds using multiple rooted dummy point formulation. In the LB-column italics shows at which iteration a feasible optimal solution was found, while in the UB-column it shows at which iteration our heuristic found the optimal solution. The t-column gives total run time.

Bounds with multiple roots, and edge reduction-heuristic

	Edges	UB	z_{opt}/UB	t
S32SF20	31	34.4420	1	0.1
S64SF20	64	59.4580	0.9986	0.1
S128SF20	127	104.7950	1	0.2
S187SF20	186	141.9912	1	0.2
S32SF200	40	133.4277	1	0.1
S64SF200	73	187.52	0.9976	0.1
S128SF200	144	302.117	0.9548	0.2
S187SF200	205	389.497	0.9947	0.3
P100.1	115	1113150	1	0.5
P200	218	1391126	0.9716	0.1
P400	451	2640133	0.9883	64
K400	1026			600

Table 8.5: Results from the edge reduction-heuristic. Number of edges left is given in Edges. UB give the best integer bound found. Maximum time is set to 600 seconds. This only includes the time used in Xpress Mosel. Mosel did not find an integer solution for K400.

8.1.3 Maximum number of nodes

In Table 8.6 we give results for the dummy point formulation with the additional constraint that only 9 nodes may be part of the corresponding PCSTP-solution. We use the optimal value as initial upper bound, as we have not created a heuristic approach for this problem. The SF200 instances all have less than 9 nodes in the solution, and give the same value as in Table 8.1. We get fairly strong bounds.

Bounds with maximum nodes constraint, UB = $\mathbf{z_{opt}}$

	LB	z_{opt}	LB/z_{opt}	t
S16SF20	17.9468	18.1439	0.9891	0.4
S32SF20	36.8194	37.2120	0.9894	0.6
S64SF20	88.9775	89.8205	0.9906	1.3
S128SF20	193.7152	194.5636	0.9956	3.7
S187SF20	295.1895	296.1734	0.9967	7.8
P100.1	1217547	1249047	0.9748	1.3
P200	1917469	1958305	0.9791	2.4
P400	3888340	3949160	0.9846	7.1
K400	327202	350093	0.9346	7.4

Table 8.6: Bounds with maximum 9 nodes in the optimal solution. Initial upper bound equal to optimal value.

8.2 Flow formulations

In this section we give results for the Lagrangian relaxation of Flow Formulation 2. We give dual values and run time for all the stomach cancer instances in Table 8.7, and show plots of typical dual values during the solution process in Figure 8.4.

We see that we get strong bounds for the stomach cancer test instances when we have a high scaling factor, and weak bounds with a low scaling factor. The main difference in these two sets of test instances, is that with SF20 almost all vertices are part of the optimal solutions, while for SF200 only a small amount of vertices are part of the optimal solutions. As the subproblem consist of finding shortest paths between pairs of vertices, it is possible that with fever vertices in an optimal solution, the shortest paths will easier resemble the optimal solution.

We see that the time needed is significantly higher than for the dummy point formulation, and that most of this time is spent to solve the subproblem.

From the plots in Figure 8.4 we see the zigzagging behavior of the subgradient method. Using an MST to give an upper bound gives a much bigger initial step length for the SF200 instances, and the subgradient method uses more iterations until convergence.

We have implementations of Flow Formulation 1 and 2 in Mosel, but Mosel is only able to solve the smallest instances (S16) before running out of computer memory. No results are given from these implementations.

Dual bounds from Flow Formulation 2, UB = $\mathbf{z_{opt}}$

	SG1	SG2	z_{opt}	LB/z_{opt}	t_{SG1}	t_{SG2}	t_{GSP}
S16SF20	17.5979	17.4017	18.1439	0.9699	4	4	3
S32SF20	24.8960	24.4483	32.0602	0.7765	14	14	8
S64SF20	44.6684	44.3380	59.2132	0.7544	65	66	35
S128SF20	79.6119	79.4804	104.6270	0.7609	430	385	290
S187SF20	106.1831	106.1502	141.8218	0.7487	1200	1100	850
S16SF200	22.8100	22.7011	22.8100	1	4	4	3
S32SF200	51.8576	51.9344	51.9394	0.9999	15	14	8
S64SF200	107.3888	107.3660	108.7540	0.9874	65	66	40
S128SF200	211.4373	211.4117	213.5898	0.9899	385	360	260
S187SF200	312.2848	312.2749	316.4688	0.9868	1190	1120	850

Table 8.7: Dual bounds using Flow Formulation 2, where the optimal solution is used as upper bound. Total run time for SG1 and SG2 is given in the t_{SG1}- and t_{SG2}-columns, and the average time between SG1 and SG2 used in GraphShortestPath during the solution process is given in the t_{GSP}-column.

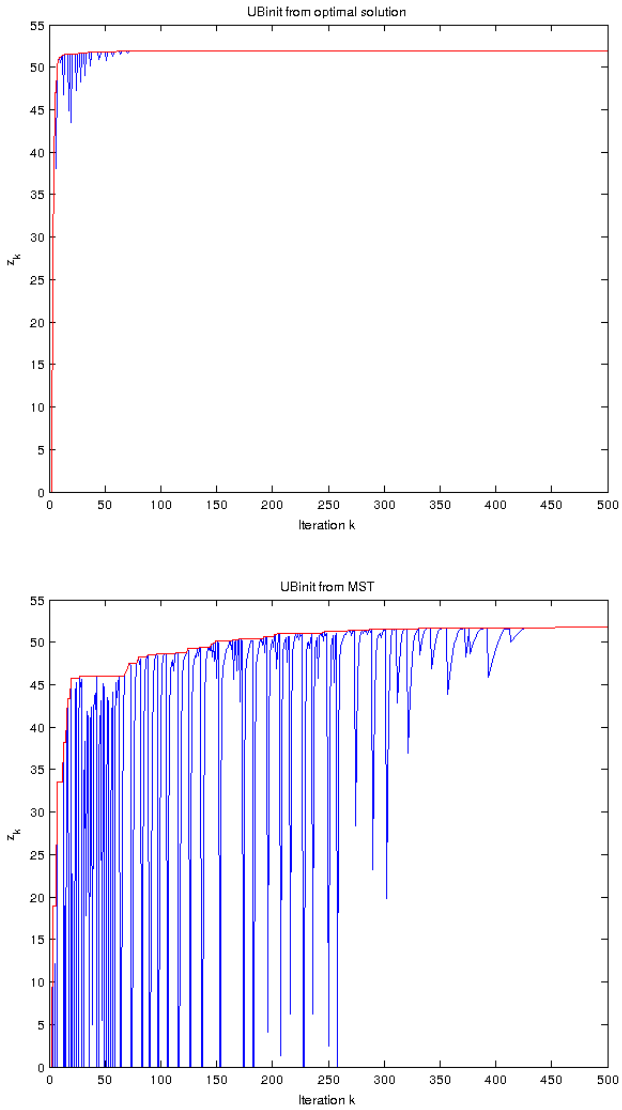

Figure 8.4: Dual bound for S32SF200. On the top we use the optimal solution as an upper bound, and for the lower one we use an MST to obtain an upper bound. Red shows lower bound and blue the dual value z_k. This shows the importance of a good initial upper bound. SG1 is used for both plots. SG2 has less zigzagging, but shows much of the same behavior. Plots for SG2 are not included.

8.3 Comparison of dual bounds

In this section we compare the dual bounds of our Lagrangian relaxations, with the LP-relaxation of the MTZ-formulation in Table 8.8-8.10. For all these tables the MTZ-column gives results for the LP-relaxation of the MTZ-formulation, the F2-column for Flow Formulation 2, and the Dummy-column for the dummy point formulation. For cases where the optimal value is known, we give LB/z_{opt}, while for the others we state LB in italics.

We see that the dummy point formulation is the strongest for the unrooted stomach cancer instances, while the results vary for the other problems. In all cases the LP-relaxation of the MTZ-formulation is fairly strong compared to the dummy point formulation.

	MTZ	F2	Dummy
S16SF20	0.9665	0.9699	0.9893
S32SF20	0.9459	0.7765	0.9757
S64SF20	0.9716	0.7544	0.9864
S128SF20	0.9845	0.7609	0.9892
S187SF20	0.9878	0.7487	0.9906
S16SF200	1	1	1
S32SF200	1	0.9999	1
S64SF200	0.9926	0.9874	0.9953
S128SF200	0.9962	0.9899	0.9967
S187SF200	0.9962	0.9868	0.9967
P100.1	0.9250		0.9287
P200	0.9644		0.9571
P400	*2377694*		*2370896*
K400	*244610*		*246303*

Table 8.8: Comparison of dual bounds using an artificial root.

	MTZ	Dummy
S16SF20	0.9665	0.9891
S32SF20	0.9862	0.9894
S64SF20	0.9861	0.9906
S128SF20	0.9944	0.9956
S187SF20	0.9961	0.9967
P100.1	0.9908	0.9748
P200	0.9867	0.9791
P400	0.9901	0.9846
K400	0.9335	0.9346

Table 8.9: Comparison of dual bounds using maximum 9 nodes in the solution.

	MTZ	Dummy
S16SF20	1	1
S32SF20	1	0.9967
S64SF20	0.9964	0.9939
S128SF20	0.9973	0.9964
S187SF20	0.9973	0.9980
S16SF200	1	1
S32SF200	0.9975	0.9967
S64SF200	0.9658	0.9609
S128SF200	0.9837	0.9621
S187SF200	0.9827	0.9677
P100.1	0.9805	0.9818
P200	0.9993	0.9932
P400	0.9912	0.9865
K400	*278686*	*280531*

Table 8.10: Comparison of dual bounds using multiple roots.

8.4 Reduction tests

In Table 8.11 we show results of the minimum adjacency test for the stomach cancer case. We see that when the cost of the edges are relatively small, more reductions can be made. For a greater scaling factor almost no reductions can be made. We only report on the number of vertices, as they are all still complete graphs. This has been used to obtain optimal solutions for the stomach cancer instances with scaling factor 20. Note that together with the dummy point formulation one can not use this reduction test.

	S16	S32	S64	S128	S187
SF20	11	12	15	17	23
SF250	16	32	64	127	186

Table 8.11: Vertices left after use of the minimum adjacency test.

8.5 Final remarks

According to Vidar Beisvåg, the optimal solution from the unrooted S187SF200 case gave realistic outputs. The set of genes were, with some exceptions, genes known to be expressed differently for diffuse cancer with signet ring cells. Due to memory and time consumption we have not been able to solve the full set of 16110 genes. Some medium sized sets have been tested, but the run time quickly increases. For test instances of 1000 genes, the run time increases to a couple of hours. Due to this we have not been able to test the multiple rooted PCSTP on the full set of genes, with interesting genes picked by Beisvåg.

The heuristics show that optimal solutions for most of the unrooted stomach cancer instances can be found by solving the PCSTP over more or less only the edges from an MST of the input graph. A heuristic simply based on reducing the edges of an input graph to the edges of its MST, and solving this to optimality by for instance the MTZ-formulation, could be tried. Together with the fast and fairly strong LP-relaxation of the MTZ-formulation, one could then quickly get guarantees on the strength of the solution. If the results of the work done here is valid for larger instances of the stomach cancer data, then these bounds are likely to be strong.

8.6 Conclusions

In this thesis we have formulated the PCSTP as several integer programs, shown Lagrangian relaxations of these IPs, and solved them using the subgradient method.

The flow based formulation required a long run time, and gave weak bounds for several of the test instances. The dummy point formulation was fairly fast and tight for most of our test instances, although a duality gap exists. The use of an artificial root gave slightly weaker bounds than looping, but this might be a result of the subgradient method. The difference was however so small, that the savings in run time make the artificial root the best choice.

Two Lagrangian heuristics have been implemented, both finding the optimal solutions of all but one of the unrooted instances where we know the optimal solution. The run time of the heuristics can be improved. We have also looked at a multiple rooted PCSTP, where the heuristics found the optimal solutions for several instances, and close to optimal solutions for the remaining instances.

Bibliography

[1] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[2] A.S. da Cunha, A. Lucena, N. Maculan, and M.G.C. Resende. A algorithm for the prize-collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 157(6):1198–1217, 2009.

[3] M.L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[4] Antonio Frangioni. About lagrangian methods in integer optimization. *Annals of Operations Research*, 139(1):163–193, 2005.

[5] M. Haouari and J. Chaouachi Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33(5):1274–1288, 2006.

[6] M. Haouari, S.B. Layeb, and H.D. Sherali. The prize collecting Steiner tree problem: models and Lagrangian dual optimization approaches. *Computational Optimization and Applications*, 40(1):13–39, 2008.

[7] M. Haouari, S.B. Layeb, and H.D. Sherali. Strength of Three MIP Formulations for the Prize Collecting Steiner Tree Problem with a Quota Constraint. *Electronic Notes in Discrete Mathematics*, 36:495–502, 2010.

[8] Markus Leitner and Günther Raidl. Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks. *Hybrid Metaheuristics*, pages 158–174, 2008.

[9] I. Ljubić, R. Weiskircher, U. Pferschy, G.W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105(2):427–449, 2006.

[10] A. Lucena and M.G.C. Resende. Strong lower bounds for the prize collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 141(1):277–294, 2004.

[11] Abilio Lucena. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140(1):375–410, 2005.

[12] J Lundgren, M Rönnqvist, and P Värbrand. Optimization. studentlitteratur. *Lund, Sweden*, 2010.

[13] CE Miller, A.W. Tucker, and RA Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[14] E. Uchoa. Reduction tests for the prize-collecting Steiner problem. *Operations research letters*, 34(4):437–444, 2006.

[15] LA Wolsey. Integer programming. *Wiley-Interscience series in discrete mathematics and optimization*, 1998.