Dynamic Programming: Salesman to Surgeon

by

David Qian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2013

Abstract

Dynamic Programming is an optimization technique used in computer science and mathematics. Introduced in the 1950s, it has been applied to many classic combinatorial optimization problems, such as the Shortest Path Problem, the Knapsack Problem, and the Traveling Salesman Problem, with varying degrees of practical success.

In this thesis, we present two applications of dynamic programming to optimization problems. The first application is as a method to compute the Branch-Cut-and-Price (BCP) family of lower bounds for the Traveling Salesman Problem (TSP), and several vehicle routing problems that generalize it. We then prove that the BCP family provides a set of lower bounds that is at least as strong as the Approximate Linear Program (ALP) family of lower bounds for the TSP. The second application is a novel dynamic programming model used to determine the placement of cuts for a particular form of skull surgery called Cranial Vault Remodeling.

Acknowledgements

I would like to thank my supervisor and mentor Ricardo Fukasawa. Without his valuable guidance, input, and time, I would not have been able to write the thesis you are reading now.

Secondly, I would like to thank my reader Jochen Könemann, who also collaborated with us on the Cranial Vault Remodeling project. I would also like to thank my second reader, Laura Sanita, for her valuable comments. Additionally, I would like to thank Nikoo Saber from The Hospital for Sick Children for her correspondence with us on the CVR project.

Finally, I would like to thank my friends and fellow graduate students who helped me with their time and advice. A special thanks goes out to Becky, Brandon, Dale, Devanshu, Leanne, and Chloe.

Dedication

I dedicate this thesis to my family, for their support, patience, time, and love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Preliminaries

Dynamic Programming is an optimization technique introduced by Richard Bellman in the 1950s, and has since grown to be widely used across computer science and operations research [19]. It is a method for solving complex problems by decomposing them into simpler, overlapping subproblems. Examples of its use include industrial applications such as production planning, patient scheduling, long-term investment programs, and inventory policies [5], and medical applications such as RNA secondary structure prediction [27] and DNA sequence alignment [22]. In addition to the classical model developed by Bellman, other flavours of dynamic programming include stochastic dynamic programming, frequently used to model animal behaviour [21], and approximate dynamic programming.

This thesis presents two different applications of classical dynamic programming to optimization problems. The first is as a tool for obtaining lower bounds for the Traveling Salesman Problem (TSP) and several vehicle routing problems that generalize it. The second is as a method to decide on the placement of cuts for a particular form of skull surgery, a project undertaken in conjunction with The Hospital for Sick Children.

In the following section we will discuss basic definitions and concepts necessary for the subject matter of this thesis. Definitions from dynamic programming and linear programming will be covered, including duality, integer programming, and column generation. Therefore, this section may be skipped by the reader if they are familiar with these subjects. We conclude by giving motivations for and an outline of this thesis.

1.1 Basic Concepts

1.1.1 Dynamic Programming

Dynamic Programming is an optimization technique that was created to ‘treat the mathematical problems arising from the study of various multi-stage decision processes’ [5]. Given a sequence of decisions to be made, dynamic programming breaks them down into simpler subproblems. In doing so, it avoids attempting to enumerate all feasible decision sequences, as even with a moderate number of decisions and a moderate number of choices at each decision, the resulting dimensionality can be infeasibly high.

For dynamic programming to be applicable to a problem, it must exhibit *optimal substructure* and *overlapping subproblems*. A problem has optimal substructure if its optimal solution can be determined by combining the optimal solutions of its subproblems. The overlapping subproblem property implies that many of these subproblems within the overall problem are repeated. Because of this, a dynamic programming algorithm can avoid repetition of work through *memoization*, the storage of subproblem solutions. When the result of a subproblem is required, and that subproblem’s solution has already been determined, its result can be recalled from memory rather than reevaluated.

We describe dynamic programming formally with the Bellman equation. We index our sequence of decisions with time t, and let x_t be the state at that time, with the initial state being x_0. Depending on which state we are in, we are presented with a set of actions $a_t \in \Gamma(x_t)$. When action a is taken, we say that the state changes from x to a new state $T(x, a)$, and that the payoff (or cost) from taking action a in state x is $F(x, a)$.

Assuming n decisions are to be made, and the objective is to maximize payoff, we wish to solve the following:

$$V(x_0) = \max_{\{a_t\}_{t=0}^{n}} \sum_{t=0}^{n} F(x_t, a_t)$$

subject to

$$a_t \in \Gamma(x_t),\ x_{t+1} = T(x_t, a_t),\ \forall\ t = 0...n$$

where $V(x_0)$ represents the optimal value obtained from the objective function.

In this form, it is not immediately apparent what strategy should be employed in solving for the optimal sequence of actions, the ‘optimal policy’. Richard Bellman’s Principle of Optimality describes how to decompose the overall problem into smaller subproblems: “An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

From this, we consider only the first decision a_0.

$$\max_{a_0}\left\{F(x_0,a_0)+\max_{\{a_t\}_{t=1}^{n}}\left(\sum_{t=1}^{n}F(x_t,a_t) : \text{s.t. } a_t\in\Gamma(x_t),\ x_{t+1}=T(x_t,a_t),\ \forall\ t=1...n\right)\right\}$$

subject to $a_0 \in \Gamma(x_0)$, $x_1 = T(x_0, a_0)$.

From here, the recursive nature of our formulation becomes clear, as to solve the smaller subproblem we start with considering the first decision again, in this case a_1. Rewriting our problem recursively, we get the following:

$$V(x_0)=\max_{a_0}\{F(x_0,a_0)+V(x_1)\}$$

subject to $a_0 \in \Gamma(x_0)$, $x_1 = T(x_0, a_0)$. This gives us the general Bellman equation for any state x:

$$V(x)=\max_{a\in\Gamma(x)}\{F(x,a)+V(T(x,a))\}$$

As an example, we present the dynamic programming formulation for the NP-Hard 0-1 Knapsack Problem [19]: Given n objects with strictly positive weights $w_1, w_2, ..., w_n$ and values $v_1, v_2, ..., v_n$, and a bag with maximum carrying weight W, we wish to determine the subset of items whose total weight is $\leq W$ and maximizes the sum of the values. In other words, we wish to solve the following:

$$\begin{aligned}\max\ & \sum_{i=1}^{n} v_i x_i\\ \text{s.t.}\ & \sum_{i=1}^{n} w_i x_i \leq W\\ & x_i\in\{0,1\} \qquad \forall\ 1\leq i\leq n\end{aligned}$$

We will demonstrate how to solve this problem using Bellman's equation by presenting a dynamic programming model. We let our states be the set of pairs (i, w), for $1 \leq i \leq n$, $0 \leq w \leq W$. In state (i, w), we wish to make the decision of earning the maximum value that can be achieved with weight less than or equal to w, using only the items 1 to i. Thus, we can say that the overall problem is to determine the set of objects that maximize the value of state (n, W), i.e. $V(n, W)$.

Assume we are in state (i, w), and consider the set of actions $\Gamma(i, w)$ that we could take from this state. We will focus on item i, and decide on the rest of the items recursively; therefore, we limit the set $\Gamma(i, w)$ only to actions regarding the item i. There are two possibilities for item i: We either include it (which we shall call action I), or exclude it (action E).

We first note that if the weight of item i is greater than our available weight, i.e. $w_i > w$, then we cannot include item i in our bag. If $w_i \leq w$, then both including it or excluding it are possibilities. Therefore,

$$\Gamma(i, w) = \begin{cases} \{E\} & : \text{if } w_i > w \\ \{I, E\} & : \text{if } w_i \leq w \end{cases}$$

We now observe the payoff and new state that arise from taking either action. Clearly, by excluding item i we gain no immediate payoff from it, so $F((i, w), E) = 0$. Also, our weight remains unchanged, and we only need to consider the items up to $i - 1$. Thus, $T((i, w), E) = (i - 1, w)$. Now consider the result if we include item i. Our payoff gained is the value of item i, so $F((i, w), I) = v_i$. Our available weight is reduced by the weight of item i, and we again only need to consider the items up to $i - 1$. Thus, $T((i, w), I) = (i - 1, w - w_i)$.

We can now describe the recursive formulation for the maximum payoff of a state (i, w), $V(i, w)$:

$$V(i, w) = \begin{cases} V(i - 1, w) & : \text{if } w_i > w \\ \max\{v_i + V(i - 1, w - w_i),\ V(i - 1, w)\} & : \text{if } w_i \leq w \end{cases}$$

where the first case is when the only possible action for i is to exclude it, and the second case is when both actions are possible. To solve this recursion efficiently, we use a table to store the values of $V(i, w)$ that have already been computed.

Dynamic programming has been used on many classic optimization problems, though the efficiency of the resulting algorithms are mixed. The Shortest Path Problem with non-negative edge weights can be solved in polynomial time using Dijkstra's Algorithm. The aforementioned Knapsack Problem can be solved in pseudo-polynomial time using dynamic programming. Dynamic programming also provides the fastest known exact algorithm for the Traveling Salesman Problem, but the runtime is exponential.

1.1.2 Linear Programming

Linear programming is the study of finding a vector x that minimizes (or maximizes) a given linear function $c^\intercal x$, subject to x satisfying a given system of linear inequalities $Ax \leq b$, where A is an $m \times n$ matrix with rational coefficients. A geometric interpretation of Linear Programming is as an optimization problem within an n-dimensional polyhedron, a (possibly empty) intersection of half-spaces defined by the constraints. This optimization problem has been heavily studied, and for many years the Simplex Method devised by Dantzig was the standard method in solving these programs. The Simplex Method operates by determining an extreme point (a vertex) of the polyhedron, and moving (pivoting) to adjacent vertices until an optimal vertex is found. More recently, Karmarkar popularized an interior-point method as a provably polynomial-time algorithm for solving LPs [10].

A linear program is typically presented as the following:

$$\begin{array}{lrl} \min & c^\intercal x & \\ \text{s.t.} & Ax & \leq b \\ & x & \geq 0 \\ & x & \in \mathbb{R}^n \end{array} \tag{P}$$

The problem (P), usually referred to as the *primal*, has a corresponding linear programming problem called the *dual*, which represents a maximization problem if (P) is a minimization problem, and vice-versa. It is typically presented as the following:

$$\begin{array}{lrl} \max & b^\intercal y & \\ \text{s.t.} & A^\intercal y & \geq c \\ & y & \geq 0 \\ & y & \in \mathbb{R}^m \end{array} \tag{D}$$

The dual is a fundamental part of the study of linear programming and convex programming in general. One important fact is that the dual of the dual is the original primal. We note that it is not necessarily true that the primal or dual is feasible; it is possible for one, or both, to have no solution. Determining if a given linear program has a feasible solution requires Farkas' Lemma.

Lemma (Farkas' Lemma for Inequalities)**.** *The system $Ax \leq b$ has a solution x if and only if there is no vector y satisfying $y \geq 0$, $y^\intercal A = 0$, and $y^\intercal b < 0$.*

With these definitions, we can describe the Duality Theorems (weak and strong).

Theorem (Weak Duality). *Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. Suppose that $\bar{x}$ is a feasible solution to $Ax \leq b$ and $\bar{y}$ is a feasible solution to $y \geq 0$, $y^\intercal A = c^\intercal$. Then*

$$c^\intercal \bar{x} \leq \bar{y}^\intercal b$$

Theorem (Strong Duality). *Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$. Then*

$$\max \{c^\intercal x : Ax \leq b\} = \min \{y^\intercal b : y \geq 0 : y^\intercal A = c^\intercal\}$$

provided that both sets are nonempty.

In other words, the objective value of a solution to a linear program is not only bounded by the value of a solution to its dual, but is in fact equivalent for optimal solutions.

1.1.3 Integer Programming

One aspect of linear programming is that the solution set is continuous and, more strongly speaking, convex. These characteristics allow for relatively fast methods to solve LPs. Conversely, *Integer Programs* (IPs) are problems where some (a *mixed* IP) or all (a *pure* IP) of the variables must take on integer values. These variables are typically used to model indivisibilities in problems (such as building a number of cars) or to model decisions by using 0/1 variables (such as facility placement) [26]. While integer programs are able to model problems that linear programs can't, solving them is NP-Hard.

An important concept in Integer Programming is that of a *relaxation*. Consider the following pure IP with linear constraints:

$$\begin{array}{rl} \min & c^\intercal x \\ \text{s.t.} & Ax \leq b \\ & x \in \mathbb{Z}^n \end{array} \tag{IP}$$

Then the following program

$$\begin{array}{rl} \min & f(x) \\ \text{s.t.} & x \in R \end{array} \tag{R}$$

is a relaxation of (IP) as long as $\{x : Ax \leq b, x \in \mathbb{Z}^n\} \subseteq R$, and $f(x) \leq c^\intercal x$ for all solutions x to (IP).

A simple example of a relaxation to (IP) is its associated linear program. Instead of the integrality requirements on x, we *relax* them to $x \in \mathbb{R}^n$.

$$\begin{array}{rl} \min & c^\intercal x \\ \text{s.t.} & Ax \leq b \\ & x \in \mathbb{R}^n \end{array} \tag{LP}$$

From this simple example we see that the optimal value for (LP) must be less than or equal to the optimal value for (IP), since the set of solutions x that satisfy the constraints of (IP) also satisfy the constraints of (LP). In fact, this must be true for *any* relaxation of a given minimization IP. Thus, the optimal value of a relaxation provides a lower (or upper) bound on the optimal value of a minimization (or maximization) IP. Lower and upper bounds to integer programs are essential for Branch-and-Bound, an algorithm commonly used to solve integer programs.

Given this broad definition of relaxations, we focus our interest on *useful* relaxations. Generally speaking, we want a relaxation to be quick and easy to solve with respect to the original problem, and for the lower (or upper) bounds to be as close to the optimal solution as possible.

Relaxations of an IP may come, for instance, as projections of higher dimensional sets. Consider (IP), and let $P = \{x : Ax \leq b\}$ be the set of points described by its constraints, excluding the integrality requirements. We can then write (IP) as follows:

$$\begin{array}{ll} \min & c^{\intercal}x \\ \text{s.t.} & x \in P \cap \mathbb{Z}^n \end{array} \tag{P-IP}$$

Now, consider the following set R:

$$R = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^m : Bx + Cy \leq b\}$$

for some matrices B, C, and vector b. Using this notation, R is a (higher-dimension) relaxation of (P-IP) if the following is true:

$$\{x \in \mathbb{Z}^n : x \in P\} = \{x \in \mathbb{Z}^n : x \in \mathbf{Proj}_x(R)\} \subseteq \mathbf{Proj}_x(R)$$

where $\mathbf{Proj}_x(R)$ is defined as the following [1]:

$$\mathbf{Proj}_x(R) := \{x \in \mathbb{R}^n : \exists y \in \mathbb{R}^m : (x, y) \in R\}$$

Two different relaxations can be compared by considering their bounds and feasible regions. Consider two relaxations of (P-IP), R_1 and R_2. We say that R_1 is a *tighter* relaxation than R_2 if the following holds:

$$\mathbf{Proj}_x(R_1) \subseteq \mathbf{Proj}_x(R_2)$$

Given that R_1 is tighter than R_2, this leads to the following:

$$\min_{x \in \mathbf{Proj}_x(R_1)} c^{\intercal}x \geq \min_{x \in \mathbf{Proj}_x(R_2)} c^{\intercal}x$$

1.1.4 Column Generation

Column Generation is a technique for solving a mathematical program by iteratively adding the variables of the model, introduced by Ford *et al.* [12]. When the number of variables dominates the number of constraints, most variables are non-basic in an extreme solution, and have no contribution to the optimal solution. Thus, only a small fraction of variables are needed to prove optimality [20]. Column Generation is typically used for solving linear programs with huge numbers of variables.

We would like to solve a linear program, called the *master problem* (MP),

$$\begin{aligned} \min \quad & \sum_{j \in J} c_j \lambda_j \\ \text{s.t.} \quad & \sum_{j \in J} \mathbf{a}_j \lambda_j \geq \mathbf{b} \\ & \lambda_j \geq 0, \; j \in J \end{aligned} \tag{MP}$$

with $|J| = n$ variables and m constraints. In many applications, n is exponential in terms of m, and working with the full set J is not an option due to its size. Instead, we try to determine the optimal solution of the *restricted master problem* (RMP), which contains a subset $J' \subseteq J$ of variables:

$$\begin{aligned} \min \quad & \sum_{j \in J'} c_j \lambda_j \\ \text{s.t.} \quad & \sum_{j \in J'} \mathbf{a}_j \lambda_j \geq \mathbf{b} \\ & \lambda_j \geq 0, \; j \in J' \end{aligned} \tag{RMP}$$

The challenge is determining when an optimal solution to the RMP is also an optimal solution to the MP. Consider an optimal dual solution π^* of the RMP. In column generation, we look for a nonbasic variable of negative reduced cost, by solving the *pricing problem* (PP):

$$\min \{c_j - \pi^* \mathbf{a}_j : j \in J\} \tag{PP}$$

When this value is < 0, the variable λ_j and its coefficients c_j, $\mathbf{a}_j$ corresponding to a minimizer j are added to the RMP. Iterating, we again solve the RMP to optimality and use the dual variables to determine if there are any new variables with negative reduced cost. Once there are no new variables to be added, an optimal solution to the RMP is optimal for the MP as well.

In general, solving (PP) by inspecting every element of J might not be computationally feasible when J is very large. However, in many applications the entities of J can be described as the feasible domain X of an optimization problem, so that the pricing problem can be solved as

$$\min_{\mathbf{x}\in X} \{c(\mathbf{x}) - \pi^* \mathbf{a}(\mathbf{x})\}$$

Thus, rather than having to explicitly enumerate through J, the pricing problem can be solved through a structured optimization problem.

1.2 Motivations and Outline

In Chapter 2, we discuss two different lower bounds on the Traveling Salesman Problem: the Branch-Cut-and-Price (BCP) model by Fukasawa *et al.* [13] in 2006, and an Approximate Linear Program (ALP) model recently presented in 2012 by Toriello [25], giving a family of bounds that were demonstrated to be greater than or equal to the Held-Karp bound. In [24], Toriello shows that a particular member of the family of ALP bounds is equivalent to the Held-Karp bound.

The BCP bound is obtained through both column generation by using dynamic programming, and cut generation. It has shown strong computational success in solving previously intractable instances of problems which generalize TSP, such as Capacitated Vehicle Routing, Capacitated Minimum Spanning Tree, Generalized Assignment, Time Dependent Traveling Salesman, and Capacitated Arc Routing. A natural question to consider is which of the two families of LPs (ALP or BCP) gives a better bound on the TSP, and if one dominates the other. We answer this question by demonstrating that, given a solution to the BCP model, we can construct a feasible solution to the ALP model with the same objective value. This shows that the BCP bound is at least as strong as the ALP bound.

In Chapter 3, we shift direction to present a medical application of dynamic programming. We briefly discuss Cranial Vault Remodeling. This form of surgery is undertaken on children experiencing a premature fusing of their soft skull bones. The surgery involves removing a piece of the front of the skull from the patient and reshaping it into an appropriate curvature by cutting and remodelling the bone. The current methodologies require the artistic judgment of the craniofacial surgeon [6], which can lead to opportunities of error by less experienced surgeons. Recent advances by The Hospital for Sick Children introduced the notion of an ideal skull curvature that can be generated for a given patient. Using this ideal skull curvature, a metal 'bandeau' is machined to guide the surgeon. The question still arises of where the surgeon should cut the removed skull bone in order to match the bandeau.

Using CT scans of patients' skulls and the bandeau templates, we have developed a model to determine where the frontal bone should be cut in order to attain an ideal curvature post-surgery. We present our solutions to the problem, including an Integer Programming model and a Dynamic Programming model, the latter of which resulted in a polytime algorithm for solving the problem. Our methods are currently under evaluation by The Hospital for Sick Children, with very positive initial feedback.

Chapter 2

Dynamic Programming based-Bounds for Routing Problems

2.1 Traveling Salesman Problem

The Traveling Salesman Problem is a classic NP-Hard problem [10]. Its general description is as follows [10]: Given a set of cities and distances between every pair of cities, the 'salesman' wishes to visit every city exactly once, and then return to his starting city, in the shortest possible total distance. The problem has been the focus of heavy study, and many heuristics and techniques exist to allow instances of several thousand cities to be solved to optimality.

For this thesis, we will consider a version of the TSP where we use a directed, complete graph. Let $G = (V, A)$ be the complete digraph on vertices $V = \{0\} \cup N$, where $N = \{1, 2, ..., n\}$. Without loss of generality, let 0 be the salesman's starting point, which we will call the depot. Each directed arc ij has an associated non-negative cost, c_{ij}. We make the arc completeness assumption to simplify notation; however, this assumption on the graph is not required for our results.

In terms of exact algorithms, a Dynamic Programming model gives the best known running time. The TSP can be converted into a shortest path problem on a network that has an exponential number of nodes with respect to the number of cities. Given non-negative distances, this shortest path problem can be solved using Dynamic Programming. However, the runtime is $O(n^2 2^n)$ [3], exponential with respect to the inputs. Additionally, the DP solution has exponential space requirements. The details of this model are explained in section 2.3

The Traveling Salesman Problem has been described using an Integer Programming formulation that dates back as far as the 1950s from the work of Dantzig *et al.* [11]. The binary integer formulation is as follows:

$$\min \sum_{i \in N} (c_{0i} x_{0i} + \sum_{j \in N \setminus i} c_{ij} x_{ij} + c_{i0} x_{i0}) \tag{2.1}$$

$$\text{s.t.} \sum_{i \in N} x_{0i} = \sum_{i \in N} x_{i0} = 1 \tag{2.2}$$

$$x_{0i} + \sum_{j \in N \setminus i} x_{ji} = \sum_{j \in N \setminus i} x_{ij} + x_{i0} = 1 \qquad \forall\, i \in N \tag{2.3}$$

$$\sum_{i \in U} (x_{0i} + \sum_{j \in N \setminus U} x_{ji}) \geq 1 \qquad \forall\, \emptyset \subsetneq U \subsetneq N \tag{2.4}$$

$$x_{ij} \in \mathbb{Z}_+ \qquad \forall\, (i,j) \in A \tag{2.5}$$

In IP form, the variable x_{ij} represents the salesman using arc ij in a tour. The objective function (2.1) aims to minimize the cost of each edge used in the chosen tour. Constraint (2.2) represents the tour starting and ending at the depot, never visiting it in-between. Constraint (2.3) is the city flow constraint; each city is entered exactly once, and left exactly once. Constraint (2.4) is the subtour elimination constraint; for any strict subset of cities, there can be no cycles within it (i.e. no subtours). In other words, any strict subset of cities must have an arc leaving it and an arc entering it, so that the resulting graph is strongly connected.

A simple relaxation of the IP formulation is obtained by replacing the integer requirement (2.5) with the relaxation $0 \leq x_{ij} \leq 1$. The lower bound that this heavily studied LP gives is known as the Held-Karp bound [16], a value that we denote as z_{HK}.

2.2 Branch-Cut-and-Price

The Capacitated Vehicle Routing Problem (CVRP) is an NP-Hard problem that generalizes the TSP. Until recently, it had been particularly resistant computationally, despite theoretical advances. In 2006, Fukasawa *et al.* [13] introduced the Branch-and-Cut-and-Price (BCP) algorithm that produced considerable success for the CVRP. We will look at the BCP algorithm in the context of the TSP.

A possible way to approach the Traveling Salesman Problem is the set partitioning formulation. Rather than selecting edges to form a feasible tour, we consider the set of directed Hamiltonian cycles, and pick the one with the lowest cost. Let Q be the set of such tours, and let c_q be the sum of the arc costs of a tour $q \in Q$. Using these definitions, we get the following formulation:

$$\begin{aligned} \min \quad & \sum_{q \in Q} c_q \lambda_q \\ \text{s.t.} \quad & \sum_{q \in Q} \lambda_q = 1 \\ & \lambda_q \in \{0, 1\} \end{aligned}$$

Even solving the LP relaxation of this formulation is NP-Hard, since pricing the exponential number of variables is equivalent to actually solving a TSP. Instead, we relax the conditions on Q to be a set of 'q-routes', defined as the following:

Definition. *Given the graph $G = (\{0\} \cup N, A)$, a **q-route** q of G is a sequence of vertices $v_q(0), v_q(1), ..., v_q(n+1)$ such that $v_q(0) = v_q(n+1) = 0$, and $v_q(i) \in N,\ \forall\ i = 1...n$.*

Informally, a q-route is a directed walk with $n+1$ edges that begins and ends at the depot 0, and does not visit it in-between. In addition to this tour relaxation, we relax the integrality condition on λ_q to $0 \leq \lambda_q \leq 1$. Any solution to the relaxation is thus a convex combination of these q-routes. To connect the λ variables to the notion of graph edge variables, we introduce the coefficient d_{ij}^q:

$$d_{ij}^q := \sum_{k=1}^{n+1} \mathbb{1}_{\{v_q(k-1)=i, v_q(k)=j\}} \qquad \forall\ i, j \in N,\ q \in Q$$

In other words, d_{ij}^q is the number of times a q-route q uses the arc ij. Using this coefficient, we can use the λ_q variables to set arc variables x_{ij} in order to enforce in-degree, out-degree, and subtour elimination constraints as in the Held-Karp relaxation. We call the resulting formulation *BCP*, and present it as follows:

$$\min \quad \sum_{i \in N} (c_{0i} x_{0i} + \sum_{j \in N \setminus i} c_{ij} x_{ij} + c_{i0} x_{i0}) \tag{2.6}$$

$$\text{s.t.} \quad x_{ij} = \sum_{q \in Q} d_{ij}^q \cdot \lambda_q \qquad \forall\, i, j \in N \tag{2.7}$$

$$\sum_{i \in N} x_{0i} = \sum_{i \in N} x_{i0} = 1 \tag{2.8}$$

$$x_{0i} + \sum_{j \in N \setminus i} x_{ji} = \sum_{j \in N \setminus i} x_{ij} + x_{i0} = 1 \qquad \forall\, i \in N \tag{2.9}$$

$$\sum_{i \in U} (x_{0i} + \sum_{j \in N \setminus U} x_{ji}) \geq 1 \qquad \forall\, \emptyset \subsetneq U \subsetneq N \tag{2.10}$$

$$0 \leq \lambda_q \leq 1 \qquad \forall\, q \in Q \tag{2.11}$$

Though computationally this is not the most practical approach for the TSP, a similar idea introduced by Fukasawa *et al.* [13] has been very successfully applied to TSP generalizations such as Capacitated Vehicle Routing, Capacitated Minimum Spanning Tree, Generalized Assignment, Time Dependent Traveling Salesman, and Capacitated Arc Routing, which is our motivation for its use.

The authors of [13] showed that the pricing problem for the dual variables can be solved in polynomial time, and thus the LP primal can be solved in polynomial time as well. In each column generation iteration, the reduced cost of each edge, $\bar{c}_{ij}$, is calculated from the dual variables of the Restricted Master Problem. Let α, γ, and ω be the dual variables associated with constraints (2.8), (2.9), and (2.10) respectively. The reduced cost $\bar{c}_{ij}$ of an edge ij is given by:

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \gamma_i - \gamma_j - \displaystyle\sum_{\substack{U \subsetneq N \\ \delta(U) \ni ij}} \omega_U & : i, j \in N \\ c_{ij} - \alpha - \gamma_j - \displaystyle\sum_{\substack{U \subsetneq N \\ \delta(U) \ni ij}} \omega_U & : i = 0,\, j \in N \end{cases}$$

The pricing subproblem of finding the q-routes (columns) to add to the RMP can be solved by using an $(n+1) \times n$ matrix M. Each entry, $M(k, i)$, represents the least costly walk that reaches vertex i from vertex 0 using exactly k steps. The entry contains a label consisting of three elements: the vertex identifier (i), the current cost of the walk $\bar{c}(M(k, i))$, and a pointer to a label representing the walk up to the previous vertex.

Initially, the only known label, $M(0,0)$, represents an empty walk and has cost zero, with all other labels initialized to be empty walks with infinite cost. From $M(0,0)$, we use dynamic programming to populate the matrix, starting with lower step values of k. For each row k, the algorithm goes through each entry i and, for each neighbour j of i, evaluates the extension of the walk represented by $M(k,i)$ to j. If $\bar{c}(M(k,i)) + \bar{c}_{ij} \leq \bar{c}(M(k+1,j))$, then $M(k+1,j)$ is updated. Formally,

$$\bar{c}(M(k,i)) = \begin{cases} \min\limits_{j \in N \setminus i} \{\bar{c}(M(k-1,j)) + \bar{c}_{ji}\} & : \text{if } i \in N,\ 1 < k \leq n, \text{ or } i = 0,\ k = n+1, \\ \bar{c}_{0i} & : \text{if } k = 1. \end{cases}$$

where $M(k,i)$ is the most negative q-route of length k that ends at vertex i.

Eventually, we will have the most negative walk with length exactly n that arrives at each vertex i. Extending the walk to the depot, we obtain the corresponding q-route. All negative q-routes found this way are added to the RMP. There are $n^2 + n$ entries in the matrix, and each is processed in $O(n)$ time, so the runtime of the pricing problem is $O(n^3)$.

The authors of the same paper also demonstrated a method to strengthen the formulation through cycle elimination. Since eliminating cycles entirely from the q-routes is NP-Hard, they settle for 't-cycle-free' q-routes for small values of t, where a walk is t-cycle-free if it contains no cycles of length t or less, including 2-cycles (directed walks of the form $i - j - i$). Formally, we say that a q-route q is t-cycle free if the vertices $\{v_q(k), v_q(k+1), ..., v_q(k+t)\}$ are all distinct for every $0 \leq k \leq n - t + 1$.

The pricing algorithm for t-cycle-free q-routes operates as previously described, except with additional memo-ization by keeping track of a bucket of labels for each entry in M. When $t > 2$, Irnich *et al.* [17] showed that using a simple labeling rule would result in a runtime bounded by $O(n^{t+2})$. A much more complex labeling rule improves this bound to a runtime of $O(t \cdot t!^2 \cdot n^3)$, polynomial for fixed t. In practice, t is usually at most 4. The reader is referred to [13] and [17] for additional details. Using this strengthening, we can replace the set Q of q-routes with the set Q^t of t-cycle-free q-routes, for $t \geq 1$, noting that $Q^1 = Q$. We call its associated LP relaxation the BCP_t formulation.

2.3 Approximate Linear Program for TSP

Recently, Toriello [25] introduced a method for constructing lower bounds for the TSP that he later showed were equivalent to the Held-Karp bound [24]. Before presenting his formulation, we first discuss the dynamic programming formulation for the TSP.

The DP formulation for the TSP uses the observation that when the salesman is at a city i, the only information required to make the decision of which city to visit next is the subset $U \subseteq N$ of cities that has not yet been visited [4]. This pair, (i, U), can be considered as a state that has unique cost (the minimum cost to visit the vertices of U, starting from i, and then return to the depot 0). The set $S = \{(i, U) : i \in N,\ U \subseteq N \setminus i\} \cup \{(0, N), (0, \emptyset)\}$ denotes every possible state. Using this set of states, we can create a 'shortest path' formulation; we start at the node $(0, N)$, and wish to end at the node $(0, \emptyset)$. Our arc set is as follows:

$$\begin{aligned} A = \quad & \{((0, N), (i, N \setminus i)) : i \in N\} \cup \{((i, \emptyset), (0, \emptyset)) : i \in N\} \\ & \cup\{((i, U \cup j), (j, U)) : i \in N, j \in N \setminus i, U \subseteq N \setminus \{i, j\}\} \end{aligned}$$

where the cost of using any arc $((i, U \cup j), (j, U)) \in A$ is the cost of the edge c_{ij}, and the costs of $((0, N), (i, N \setminus i))$ and $((i, \emptyset), (0, \emptyset))$ are c_{0i} and c_{i0} respectively.

Using this formulation, the TSP has been changed to solving the shortest path problem for the graph (S, A), which can be done with dynamic programming. This formulation is not straightforward to solve, however, due to the exponential number of nodes in (S, A) with respect to our original graph G.

This shortest path problem can be formulated as a linear program. The primal is

$$\begin{array}{lll} \min & \displaystyle\sum_{i \in N} \left(c_{0i} x_{(0,N),(i,N\setminus i)} + \sum_{U \subseteq N \setminus i} \sum_{j \in U} c_{ij} x_{(i,U),(j,U\setminus j)} + c_{i0} x_{(i,\emptyset),(0,\emptyset)} \right) & \\ \text{s.t.} & \displaystyle\sum_{i \in N} x_{(0,N),(i,N\setminus i)} = 1 & \\ & \displaystyle -x_{(0,N),(i,N\setminus i)} + \sum_{j \in N \setminus i} x_{(i,N\setminus i),(j,N\setminus\{i,j\})} = 0, & \forall\ i \in N \\ & \displaystyle -\sum_{k \in N \setminus \{U \cup i\}} x_{(k,U\cup i),(i,U)} + \sum_{j \in U} x_{(i,U),(j,U\setminus j)} = 0, & \forall\ i \in N,\ U \subseteq N \setminus i \\ & \displaystyle -\sum_{k \in N \setminus i} x_{(k,i),(i,\emptyset)} + x_{(i,\emptyset),(0,\emptyset)} = 0, & \forall\ i \in N \\ & \displaystyle\sum_{i \in N} x_{(i,\emptyset),(0,\emptyset)} = 1, & \forall\ i \in N \\ & x_a \geq 0,\ \forall\ a \in A. & \end{array}$$

As mentioned, explicitly solving this LP is not computationally feasible due to its size, and is equivalent to solving the TSP. The dual of the shortest path formulation is

$$\begin{array}{lll} \max & y_{0,N} - y_{0,\emptyset} & \\ \text{s.t.} & y_{0,N} - y_{i,N\setminus i} \leq c_{0i} & \forall\, i \in N \\ & y_{i,U\cup j} - y_{j,U} \leq c_{i,j} & \forall\, i,j \in N,\ U \subseteq N \setminus \{i,j\} \\ & y_{i,\emptyset} - y_{0,\emptyset} \leq c_{i0} & \forall\, i \in N \\ & y_{0,N}, y_{0,\emptyset} \in \mathbb{R} & \\ & y_{i,U} \in \mathbb{R} & \forall\, i \in N,\ U \subseteq N \setminus i \end{array}$$

We notice that due to the fact that the variables are free, and every constraint plus the objective function involves a pair of variables with opposing signs, given any solution y to the dual, a translation of each component by the same constant c would still result in a feasible solution of the same cost. Thus, without loss of generality we can set $y_{0,\emptyset}$ to 0, and remove it from our dual. This dual can be interpreted as a pricing problem for a tolled shuttle service; a shuttle company wishes to offer rides between the cities of $\{0\} \cup N$ that the salesman can use in lieu of his own transportation. The company wants to maximize their own profit, and the toll between two cities, $y_{i,U\cup j} - y_{j,U}$ can depend not only on the cities i and j, but the remaining set of cities U that the salesman wishes to visit afterwards. The toll that the company charges cannot be greater than the salesman's own cost, c_{ij}, between these two cities. The variable $y_{i,U}$ can thus be viewed as a 'cost-to-go'; the maximum cost that the shuttle service can charge the salesman from city i when he still must visit U.

While solving this dual LP is still NP-Hard, Toriello [24] proposed approximating the variables $y_{i,U}$ and solving the *Approximate Linear Program* (ALP) in order to obtain a dual-feasible solution (and thus a bound on the optimal TSP value). First, we pick a parameter $t \in \mathbb{Z}$, such that $0 \leq t \leq \frac{n+1}{2}$, and replace our variables with the following assignment:

$$y_{i,U} = \pi_{i,\emptyset} + \sum_{k\in U} \pi_{i,k} + \sum_{\substack{W\subseteq U \\ |W|\geq n-t}} \lambda_{i,W} + \sum_{\substack{W\subseteq N\setminus(U\cup i) \\ |W|\geq n-t}} \mu_{i,W}$$

The variable $\pi_{i,\emptyset}$ is a cost associated with being at the vertex i. The variable $\pi_{i,k}$ represents the cost of being at i and still having to visit city k. $\lambda_{i,W}$ is a state entrance toll; given that we are at city i and still need to visit $W \subseteq U$, a toll needs to be paid for entering that set of cities W. Similarly, $\mu_{i,W}$ is a state exit toll. The parameter t determines the granularity of our approximation; when t is 0, we do not consider any state entrance or exit tolls. When the cardinality constraints are removed, we consider the state tolls of every possible subset of cities $W \subseteq U$, which Toriello [25] showed is equivalent to solving the original dual LP.

After this variable replacement, the approximation becomes the following dual ALP_t:

$$\begin{aligned}
\max \quad & y_{0,N} \\
\text{s.t.} \quad & y_{0,N} - \pi_{i,\emptyset} - \sum_{k \in N \setminus i} \pi_{i,k} - \sum_{\substack{U \subseteq N \setminus i \\ |U| \geq n-t}} \lambda_{i,U} && \leq c_{0i} \\
& \quad \forall\, i \in N \\
& \pi_{i,\emptyset} + \sum_{k \in U \cup j} \pi_{i,k} + \sum_{\substack{W \subseteq U \cup j \\ |W| \geq n-t}} \lambda_{i,W} + \sum_{\substack{W \subseteq N \setminus (U \cup \{i,j\}) \\ |W| \geq n-t}} \mu_{i,W} \\
& \quad -\pi_{j,\emptyset} - \sum_{k \in U} \pi_{j,k} - \sum_{\substack{W \subseteq U \\ |W| \geq n-t}} \lambda_{j,W} - \sum_{\substack{W \subseteq N \setminus (U \cup j) \\ |W| \geq n-t}} \mu_{j,W} && \leq c_{ij} \\
& \quad \forall\, i, j \in N,\ U \subseteq N \setminus \{i, j\} \\
& \pi_{i,\emptyset} + \sum_{\substack{U \subseteq N \setminus i \\ |U| \geq n-t}} \mu_{i,U} && \leq c_{i0} \\
& \quad \forall\, i \in N
\end{aligned} \tag{ALP$_t$}$$

It was shown in [25] that the separation problem for ALP_t requires $O(n^{t+2} + n^3)$ arithmetic operations, and therefore ALP_t is solvable in polynomial time for fixed t. Toriello showed in [24] that when $t = 0$, the approximate linear program's optimal objective value is equivalent to that of the Held-Karp bound.

From this dual approximation, we can generate the primal of ALP_t. The constraints of the primal can be considered in two classes: Those associated with the π variables of the dual, and those associated with the λ and μ variables of the dual. We will first present the part of the primal LP corresponding to the objective function and π dual variables, with an implicit reference to the constraints associated with λ and μ. The reason for presenting these constraints separately is due to the nature of the λ and μ variables in the dual based on our choice of t: When t increases, new λ and μ variables are admitted into the approximation, whereas the π variables are always present regardless of the choice of t.

The primal is then:

$$\min \sum_{i\in N}\left(c_{0i}x_{0i} + \sum_{j\in N\setminus i}\sum_{U\subseteq N\setminus\{i,j\}} c_{ij}x_{ij}^U + c_{i0}x_{i0}\right) \tag{2.12}$$

$$\text{s.t. } \sum_{i\in N} x_{0i} = 1 \tag{2.13}$$

$$-x_{0i} + \sum_{j\in N\setminus i}\sum_{U\subseteq N\setminus\{i,j\}} (x_{ij}^U - x_{ji}^U) + x_{i0} = 0, \quad \forall\ i \in N \tag{2.14}$$

$$-x_{0i} + \sum_{U\subseteq N\setminus\{i,j\}} x_{ij}^U + \sum_{k\in N\setminus\{i,j\}}\sum_{U\subseteq N\setminus\{i,j,k\}} (x_{ik}^{U\cup j} - x_{ki}^{U\cup j}) = 0 \quad \forall\ i,j \in N \tag{2.15}$$

$$x \geq 0 \tag{2.16}$$

$$\text{And constraints (2.17), (2.18)}$$

Constraint (2.13) represents outflow from the depot. Constraint (2.14) states that any in-flow and out-flow of any city i must be balanced, whether that flow comes from a different city or from the depot. Constraint (2.15) has been termed an "$i-j$" flow balance requirement; the flow into i when j must still be visited must be equal to the flow out of i when j still remains, either directly to city j or to any other city.

The second class of constraints depends on the choice of t. For any city i, there is a variable in the dual, $\lambda_{i,W}$ and $\mu_{i,W}$ for each subset $W \subseteq N \setminus i$, such that $|W| \geq n - t$. These variables have the following associated constraints in the primal:

$$\lambda_{i,W}: \quad -x_{0i} + \sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ U\cup j\supseteq W}} x_{ij}^U - \sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ U\supseteq W}} x_{ji}^U = 0 \tag{2.17}$$

$$\mu_{i,W}: \quad x_{i0} + \sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup\{i,j\})\supseteq W}} x_{ij}^U - \sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup i)\supseteq W}} x_{ji}^U = 0 \tag{2.18}$$

2.4 Proof of Dominance

Given these two relaxations ALP_t and BCP_t, a natural question is to ask which one provides a stronger bound on the TSP. In this section, we show that, for any $0 \leq t \leq \frac{n+1}{2}$, BCP_{t+1}

is stronger than ALP_t. We do so by taking any solution to BCP_{t+1} and use it to create a solution to ALP_t that has the same objective value and satisfies its constraints. We note that BCP_t is only defined for $t \geq 1$, while ALP_t is for $t \geq 0$. Additionally, the complexities of solving these two formulations are comparable.

Recall the definition of d_{ij}^q; it is the number of times that a q-route q crosses arc ij. Since each q-route has exactly $n+1$ edges, we can define $d_{ij}^{q,k}$, with $1 \leq k \leq n+1$, as the following binary coefficient:

$$d_{ij}^{q,k} := \mathbb{1}_{\{v_q(k-1)=i, v_q(k)=j\}} \qquad \forall\, i, j \in N,\ q \in Q_t$$

Since each route has exactly $n+1$ edges, we get the following identity:

$$\sum_{k=1}^{n+1} d_{ij}^{q,k} = d_{ij}^q \tag{2.19}$$

Thus, the BCP_t program can be written as the following:

$$\begin{aligned}
\min \quad & \sum_{i \in N} (c_{0i} x_{0i} + \sum_{j \in N \setminus i} c_{ij} x_{ij} + c_{i0} x_{i0}) && && (2.20)\\
\text{s.t.} \quad & x_{ij} = \sum_{q \in Q^t} \left(\sum_{k=1}^{n+1} d_{ij}^{q,k} \right) \cdot \lambda_q && \forall\, i, j \in N && (2.21)\\
& \sum_{i \in N} x_{0i} = \sum_{i \in N} x_{i0} = 1 && && (2.22)\\
& x_{0i} + \sum_{j \in N \setminus i} x_{ji} = \sum_{j \in N \setminus i} x_{ij} + x_{i0} = 1 && \forall\, i \in N && (2.23)\\
& \sum_{i \in U} (x_{0i} + \sum_{j \in N \setminus U} x_{ji}) \geq 1 && \forall\, \emptyset \subsetneq U \subsetneq N && (2.24)\\
& 0 \leq \lambda_q \leq 1 && \forall\, q \in Q^t && (2.25)
\end{aligned}$$

where Q^t is the set of q-routes without t-cycles. Using this formulation, we will prove our main result:

Theorem 1. *Let $0 \leq t \leq \frac{n+1}{2}$. Given a solution $(\hat{x}, \hat{\lambda})$ to the BCP_{t+1} relaxation, it is possible to construct a feasible solution to the primal of ALP_t with the same objective value.*

This leads to the following corollary about the optimal value for the BCP relaxation $z_{BCP_{t+1}}$, and the optimal value for the ALP relaxation, z_{ALP_t}:

Corollary 1. $z_{BCP_{t+1}} \geq z_{ALP_t}$, $\forall\ 0 \leq t \leq \frac{n+1}{2}$.

To prove this theorem, we will first demonstrate how we will construct our solution to ALP_t, given a solution $(\hat{x}, \hat{\lambda})$ of BCP_{t+1}. We will first set the depot variables:

$$x_{0i} = \hat{x}_{0i} = \sum_{q \in Q^{t+1}} d_{0i}^{q,1} \cdot \hat{\lambda}_q \tag{2.26}$$

$$x_{i0} = \hat{x}_{i0} = \sum_{q \in Q^{t+1}} d_{i0}^{q,n+1} \cdot \hat{\lambda}_q \tag{2.27}$$

for all $i \in N$. The second equality in each line holds because any q-route always begins and ends at the depot 0, and never visits it in-between.

For the non-depot arc variables, consider a pair of cities $i, j \in N$. For each $U \subseteq N \setminus \{i, j\}$, such that $|U| \geq n - (t+1)$, we define the set of q-routes Q_U^{t+1}:

$$Q_U^{t+1} = \{q \in Q^{t+1} : \{v_q(1), ..., v_q(|N \setminus U|)\} = N \setminus U\}$$

In other words, if $q \in Q_U^{t+1}$, then the first $|N \setminus U|$ cities visited after the depot in q are exactly the cities in $N \setminus U$.

For each $U \subseteq N \setminus \{i, j\}$, such that $|U| \leq t+1$, we define the set $\bar{Q}_U^{t+1}$:

$$\bar{Q}_U^{t+1} = \{q \in Q^{t+1} : \{v_q(|N \setminus U| + 1), ..., v_q(n)\} = U\}$$

i.e. if $q \in \bar{Q}_U^{t+1}$, then the last $|U|$ cities visited by q before returning to the depot are exactly the cities in U.

Now, we set our variables as follows:

$$x_{ij}^U = \sum_{q \in Q_U^{t+1}} d_{ij}^{q,|N \setminus U|} \cdot \hat{\lambda}_q, \qquad \forall\ U \subseteq N \setminus \{i, j\},\ |U| \geq n - (t+1) \tag{2.28}$$

$$x_{ij}^U = \sum_{q \in \bar{Q}_U^{t+1}} d_{ij}^{q,|N \setminus U|} \cdot \hat{\lambda}_q, \qquad \forall\ U \subseteq N \setminus \{i, j\},\ |U| \leq t+1 \tag{2.29}$$

$$\sum_{\substack{U \subseteq N \setminus \{i,j\} \\ |U| = k}} x_{ij}^U = \sum_{q \in Q^{t+1}} d_{ij}^{q,|N \setminus U|} \cdot \hat{\lambda}_q, \qquad \forall\ t+1 < k < n-(t+1) \tag{2.30}$$

Lemma 1. *Let $0 \leq t \leq \frac{n+1}{2}$, and let $(\hat{x}, \hat{\lambda})$ be a solution to BCP_{t+1}. Then if we define x_{ij}^U as in (2.28), (2.29), and (2.30), the objective value of our constructed solution x for ALP_t has the same objective value as $(\hat{x}, \hat{\lambda})$ for BCP_{t+1}.*

Proof. First, we note that for a given k such that $k \leq t+1$, the sets Q_U^{t+1}, for all $|U| = n-k$, partition the q-routes in Q^{t+1}. This is because any q-route in Q^{t+1} is $(t+1)$-cycle-free, and thus the first k cities of that q-route are all unique, and thus it must fall into exactly one set Q_U^{t+1}. Therefore, we get the following identity:

$$\sum_{q \in Q^{t+1}} d_{ij}^{q,k} \cdot \hat{\lambda}_q = \sum_{\substack{U \subseteq N \\ |U| = n-k}} \sum_{q \in Q_U^{t+1}} d_{ij}^{q,k} \cdot \hat{\lambda}_q$$

Consider any set $U \subseteq N$ that contains i or j (or both). By definition, the routes of Q_U^{t+1} cannot contain both i and j in their first $n - |U|$ cities. Therefore, $d_{ij}^{q,k} = 0$ for $k \leq t+1$. Using this fact, we can exclude the sets U that contain either i or j, and extend our identity:

$$\begin{aligned}
\sum_{q \in Q^{t+1}} d_{ij}^{q,k} \cdot \hat{\lambda}_q &= \sum_{\substack{U \subseteq N \\ |U| = n-k}} \sum_{q \in Q_U^{t+1}} d_{ij}^{q,k} \cdot \hat{\lambda}_q \\
&= \sum_{\substack{U \subseteq N \setminus \{i,j\} \\ |U| = n-k}} \sum_{q \in Q_U^{t+1}} d_{ij}^{q,k} \cdot \hat{\lambda}_q \\
&= \sum_{\substack{U \subseteq N \setminus \{i,j\} \\ |U| = n-k}} x_{ij}^U \qquad\qquad \forall\ k : n-k \geq n-(t+1)
\end{aligned}$$

for all k such that $n - k \geq n - (t+1)$. Through a similar argument for $\bar{Q}_U^{t+1}$ partitioning Q^{t+1}, we can establish the same identity for k such that $n - k \leq t+1$:

$$\sum_{q \in Q^{t+1}} d_{ij}^{q,k} \cdot \hat{\lambda}_q = \sum_{\substack{U \subseteq N \setminus \{i,j\} \\ |U| = n-k}} x_{ij}^U \qquad\qquad \forall\ k : n-k \leq t+1$$

Combining this identity with our third variable setting (for all k such that $t+1 < k <$

$n-(t+1)$), we conclude that

$$\sum_{k=1}^{n+1} \sum_{\substack{U\subseteq N\setminus\{i,j\}\\ |U|=n-k}} x_{ij}^U = \sum_{k=1}^{n+1} \sum_{q\in Q^{t+1}} d_{ij}^{q,k} \cdot \hat{\lambda}_q$$
$$\sum_{U\subseteq N\setminus\{i,j\}} x_{ij}^U = \sum_{q\in Q^{t+1}} d_{ij}^{q} \cdot \hat{\lambda}_q$$
$$= \hat{x}_{ij}$$

Thus, the objective value of x in ALP_t matches that of $(\hat{x}, \hat{\lambda})$ in BCP_{t+1}. $\square$

What remains to be shown is if our constructed solution satisfies the constraints of ALP_t. To demonstrate that it does, we will prove the following two theorems:

Theorem 2. *Let* $0 \le t \le \frac{n+1}{2}$, *and let* $(\hat{x}, \hat{\lambda})$ *be a solution from* BCP_{t+1}. *Then by setting* x_{ij}^U *as in* (2.28), (2.29), *and* (2.30), *we get that* x_{ij}^U *satisfies the first constraint class* (2.13), (2.14), *and* (2.15) *for the primal of* ALP_t.

Theorem 3. *Let* $0 \le t \le \frac{n+1}{2}$, *and let* $(\hat{x}, \hat{\lambda})$ *be a solution from* BCP_{t+1}. *Then by setting* x_{ij}^U *as in* (2.28), (2.29), *and* (2.30), *we get that* x_{ij}^U *satisfies the second constraint class* (2.17) *and* (2.18) *for the primal of* ALP_t.

These two theorems, along with Lemma 1, imply our main result, Theorem 1. We start by proving Theorem 2. This proof is adapted from Toriello [24].

Proof of Theorem 2. Recall that, given our variable setting as in (2.28), (2.29), and (2.30), the following hold:

$$x_{0i} = \hat{x}_{0i}, \qquad x_{i0} = \hat{x}_{i0}, \qquad \sum_{U\subseteq N\setminus\{i,j\}} x_{ij}^U = \hat{x}_{ij}$$

for all $i \in N,\ j \in N \setminus i$. Enforcing these immediately satisfies (2.13) and (2.14) because $\hat{x}$ satisfies (2.22) and (2.23). It remains to show if the following system is feasible:

$$\sum_{k\in N\setminus\{i,j\}} \sum_{U\subseteq N\setminus\{i,j,k\}} (x_{ik}^{(U\cup j)} - x_{ki}^{(U\cup j)}) = \hat{x}_{0i} - \hat{x}_{ij}, \qquad \forall\ i \in N,\ j \in N \setminus i$$
$$\sum_{U\subseteq N\setminus\{i,j\}} x_{ij}^U = \hat{x}_{ij}, \qquad \forall\ i \in N,\ j \in N \setminus i$$
$$x \ge 0$$

Using Farkas' Lemma, this system is feasible if and only if the system

$$\sum_{i\in N}\sum_{j\in N\setminus i}((\hat{x}_{ij}-\hat{x}_{0i})\beta_{ij}-\hat{x}_{ij}\rho_{ij})<0 \tag{2.31}$$

$$\sum_{k\in U}(\beta_{ik}-\beta_{jk})+\rho_{ij}\leq 0, \qquad \forall\, U\subseteq N\setminus\{i,j\} \tag{2.32}$$

with variables β and ρ is infeasible. Given any β, we can always satisfy (2.32) by choosing small enough ρ values. To make the left-hand side of (2.31) as small as possible, we can therefore choose $-\rho_{ij}=\max_{U\subseteq N\setminus\{i,j\}}\sum_{k\in U}(\beta_{ik}-\beta_{jk})$. Thus, we can project out the ρ variables, and now need to check whether any choice of β can satisfy

$$\sum_{i\in N}\sum_{j\in N\setminus i}\left(\beta_{ij}(\hat{x}_{ij}-\hat{x}_{0i})+\hat{x}_{ij}\max_{U\subseteq N\setminus\{i,j\}}\left\{\sum_{k\in U}(\beta_{ik}-\beta_{jk})\right\}\right)<0$$

We note that the set U that is a maximizer for the terms within the parentheses can be determined greedily; if $k\in U$ is such that $(\beta_{ik}-\beta_{jk})$ is negative, we remove it from U, and if $k\notin U$ is such that $(\beta_{ik}-\beta_{jk})$ is positive, we add it to U. Because of this, we can use $(\cdot)_+ := \max\{0,\cdot\}$ and rearrange terms to rewrite this expression as

$$\begin{aligned}
&\sum_{i\in N}\sum_{j\in N\setminus i}\left(\beta_{ij}(\hat{x}_{ij}-\hat{x}_{0i})+\hat{x}_{ij}\sum_{k\in N\setminus\{i,j\}}(\beta_{ik}-\beta_{jk})_+\right)\\
=&\sum_{i\in N}\sum_{j\in N\setminus i}\left(\beta_{ij}(\hat{x}_{ij}-\hat{x}_{0i})+\sum_{k\in N\setminus\{i,j\}}\hat{x}_{ij}(\beta_{ik}-\beta_{jk})_+\right)\\
=&\sum_{i\in N}\sum_{j\in N\setminus i}(\beta_{ij}(\hat{x}_{ij}-\hat{x}_{0i}))+\sum_{i\in N}\sum_{j\in N\setminus i}\sum_{k\in N\setminus\{i,j\}}\hat{x}_{ij}(\beta_{ik}-\beta_{jk})_+
\end{aligned}$$

The condition on the last set of summations is that i,j, and k are all distinct, so we can switch the variable names for j and k, and reorder the summands:

$$\begin{aligned}
=&\sum_{j\in N}\sum_{i\in N\setminus j}(\beta_{ij}(\hat{x}_{ij}-\hat{x}_{0i}))+\sum_{j\in N}\sum_{i\in N\setminus j}\sum_{k\in N\setminus\{i,j\}}\hat{x}_{ik}(\beta_{ij}-\beta_{kj})_+\\
=&\sum_{j\in N}\sum_{i\in N\setminus j}\left(\beta_{ij}(\hat{x}_{ij}-\hat{x}_{0i})+\sum_{k\in N\setminus\{i,j\}}\hat{x}_{ik}(\beta_{ij}-\beta_{kj})_+\right)
\end{aligned}$$

The terms have been rearranged so that within the first summation, all β variables share the same second city index j. We focus on an arbitrary summand given by a fixed j and show that it is always non-negative. Without loss of generality, we can assume that $j = n$, and also assume that $\beta_{1n} \geq ... \geq \beta_{n-1,n}$. These assumptions are made for ease of notation; other cases of j and β can be proved in a similar fashion. Because of the ordering assumption on β, we can specify our bounds on k and remove the need for $(\cdot)_+$. Thus, this summand for fixed j can be written as

$$\sum_{i=1}^{n-1}\left(\beta_{in}(\hat{x}_{in} - \hat{x}_{0i}) + \sum_{k=i+1}^{n-1}\hat{x}_{ik}(\beta_{in} - \beta_{kn})\right)$$

Grouping by β terms:

$$\begin{aligned}
&= \sum_{i=1}^{n-1}\beta_{in}(\hat{x}_{in} - \hat{x}_{0i} + \sum_{k=i+1}^{n-1}\hat{x}_{ik}) - \sum_{i=1}^{n-1}\sum_{k=i+1}^{n-1}\hat{x}_{ik}\beta_{kn}\\
&= \sum_{i=1}^{n-1}\beta_{in}(\hat{x}_{in} - \hat{x}_{0i} + \sum_{k=i+1}^{n-1}\hat{x}_{ik}) - \sum_{i=1}^{n-1}\sum_{\substack{k=1\\k>i}}^{n-1}\hat{x}_{ik}\beta_{kn}\\
&= \sum_{i=1}^{n-1}\beta_{in}(\hat{x}_{in} - \hat{x}_{0i} + \sum_{k=i+1}^{n-1}\hat{x}_{ik}) - \sum_{k=1}^{n-1}\sum_{\substack{i=1\\i<k}}^{n-1}\hat{x}_{ik}\beta_{kn}
\end{aligned}$$

Swapping the k and i variable names for the last summation, we get:

$$\begin{aligned}
&= \sum_{i=1}^{n-1}\beta_{in}(\hat{x}_{in} - \hat{x}_{0i} + \sum_{k=i+1}^{n-1}\hat{x}_{ik}) - \sum_{i=1}^{n-1}\sum_{\substack{k=1\\k<i}}^{n-1}\hat{x}_{ki}\beta_{in}\\
&= \sum_{i=1}^{n-1}\beta_{in}\left(-\sum_{k=0}^{i-1}\hat{x}_{ki} + \sum_{k=i+1}^{n}\hat{x}_{ik}\right)\\
&= \beta_{1n}\left(-\hat{x}_{01} + \sum_{k=2}^{n}\hat{x}_{1k}\right) + \sum_{i=2}^{n-2}\beta_{in}\left(-\sum_{k=0}^{i-1}\hat{x}_{ki} + \sum_{k=i+1}^{n}\hat{x}_{ik}\right) + \beta_{n-1,n}\left(-\sum_{k=0}^{n-2}\hat{x}_{k,n-1} + \hat{x}_{n-1,n}\right)\\
&= \beta_{1n}\left(\sum_{k=2}^{n}(\hat{x}_{0k} + \hat{x}_{1k}) - 1\right) + \sum_{i=2}^{n-2}\beta_{in}\left(-\sum_{k=0}^{i-1}\hat{x}_{ki} + \sum_{k=i+1}^{n}\hat{x}_{ik}\right) - \beta_{n-1,n}\left(\sum_{k=0}^{n-2}(\hat{x}_{k,n-1} + \hat{x}_{kn}) - 1\right)
\end{aligned}$$

where in the last equality we use (2.22) for the leftmost sum and (2.23) for the rightmost sum, where $i = n$. Finally, we use the identity

$$\begin{aligned}
-\sum_{k=0}^{i-1}\hat{x}_{ki} + \sum_{k=i+1}^{n}\hat{x}_{ik} &= -\sum_{k=0}^{i-1}\hat{x}_{ki} + \left(-\sum_{k=0}^{i-1}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell} + \sum_{k=0}^{i-1}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell}\right) + \sum_{k=i+1}^{n}\hat{x}_{ik}\\
&= -\left(\sum_{k=0}^{i-1}\hat{x}_{ki} + \sum_{k=0}^{i-1}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell}\right) + \left(\sum_{k=0}^{i-1}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell} + \sum_{k=i+1}^{n}\hat{x}_{ik}\right)\\
&= -\sum_{k=0}^{i-1}\sum_{\ell=i}^{n}\hat{x}_{k\ell} + \sum_{k=0}^{i}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell}
\end{aligned}$$

for each $i = 2, ..., n-2$, and we can manipulate the middle sum of our last line to the following:

$$\begin{aligned}
&\sum_{i=2}^{n-2}\beta_{in}\left(-\sum_{k=0}^{i-1}\hat{x}_{ki} + \sum_{k=i+1}^{n}\hat{x}_{ik}\right)\\
=&\sum_{i=2}^{n-2}\beta_{in}\left(-\sum_{k=0}^{i-1}\sum_{\ell=i}^{n}\hat{x}_{k\ell} + \sum_{k=0}^{i}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell}\right)
\end{aligned}$$

From here, we can further manipulate our terms to separate the sums:

$$\begin{aligned}
=&\sum_{i=2}^{n-2}\beta_{in}\left(-\sum_{k=0}^{i-1}\sum_{\ell=i}^{n}\hat{x}_{k\ell} + \sum_{k=0}^{i}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell} + 1 - 1\right)\\
=&\sum_{i=2}^{n-2}\beta_{in}\left(-\sum_{k=0}^{i-1}\sum_{\ell=i}^{n}\hat{x}_{k\ell} + 1\right) + \sum_{i=2}^{n-2}\beta_{in}\left(\sum_{k=0}^{i}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell} - 1\right)\\
=&-\sum_{i=2}^{n-2}\beta_{in}\left(\sum_{k=0}^{i-1}\sum_{\ell=i}^{n}\hat{x}_{k\ell} - 1\right) + \sum_{i=2}^{n-2}\beta_{in}\left(\sum_{k=0}^{i}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell} - 1\right)
\end{aligned}$$

Using this, we can collapse our original equality into the following:

$$= \sum_{i=1}^{n-2}(\beta_{in} - \beta_{i+1,n})\left(\sum_{k=0}^{i}\sum_{\ell=i+1}^{n}\hat{x}_{k\ell} - 1\right)$$

This quantity must be non-negative because of the non-increasing assumption on the β_{in} values and the subtour elimination constraints (2.24) for $U = \{i+1, ..., n\}$ and $i = 1, ..., n-2$. Thus, by Farkas' Lemma, we have shown that the $i-j$-flow constraints (2.15) are satisfied by our variable setting. Thus, all of the constraints in the first class ((2.13), (2.14), and (2.15)) are satisfied. □

To prove Theorem 3, i.e. that we can satisfy the constraints (2.17) and (2.18) of the second class, we will need the following two lemmas:

Lemma 2. *Let* $0 \leq t \leq \frac{n+1}{2}$. *Given* $i \in N$, *and* $W \subseteq N \setminus i$, $|W| \geq n-t$, *the constraint associated with* $\lambda_{i,W}$ (2.17) *and the constraint associated with* $\mu_{i,W}$ (2.18) *in* ALP_t *can be rewritten as* $\sum_{\substack{Z \subseteq N \setminus i \\ Z \supseteq W}} \phi_{i,Z} = 0$ *and* $\sum_{\substack{Z \subseteq N \setminus i \\ Z \supseteq W}} \eta_{i,Z} = 0$ *respectively, where*

$$\phi_{i,Z} = -\sum_{j \in N \setminus (Z \cup i)} x_{ji}^{Z} + \sum_{j \in Z} x_{ij}^{Z \setminus j}$$

and

$$\eta_{i,Z} = \sum_{j \in N \setminus (Z \cup i)} x_{ij}^{N \setminus Z \cup \{i,j\}} - \sum_{j \in Z} x_{ji}^{N \setminus (Z \cup i)}$$

for all $Z \subsetneq N \setminus i$, *with a similar construction for when* $Z = N \setminus i$:

$$\phi_{i,N \setminus i} = -x_{0i} + \sum_{j \in N \setminus i} x_{ij}^{N \setminus \{i,j\}}$$

and

$$\eta_{i,N \setminus i} = x_{i0} - \sum_{j \in N \setminus i} x_{ji}^{\emptyset}$$

Lemma 3. *Let* $0 \leq t \leq \frac{n+1}{2}$. *Given a solution* $(\hat{x}, \hat{\lambda})$ *to* BCP_{t+1}, *and using our variable setting defined in* (2.28), (2.29), *and* (2.30), $\phi_{i,Z} = 0$ *and* $\eta_{i,Z} = 0$ *for all* $i \in N$, $Z \subseteq N \setminus i$, $|Z| \geq n-t$.

These two lemmas imply that the constraints (2.17) and (2.18) are satisfied, proving Theorem 3.

Proof of Lemma 2. We define $\phi_{i,Z}$, for $Z \subsetneq N \setminus i$, as the following:

$$\phi_{i,Z} = -\sum_{j \in N \setminus (Z \cup i)} x_{ji}^{Z} + \sum_{j \in Z} x_{ij}^{Z \setminus j}$$

There is a special case when $Z = N \setminus i$, for notational reasons. We define $\phi_{i,N \setminus i}$ as follows:

$$\phi_{i,N \setminus i} = -x_{0i} + \sum_{j \in N \setminus i} x_{ij}^{N \setminus \{i,j\}}$$

With these definitions, we can write out $\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}}\phi_{i,Z}$.

$$\begin{aligned}
\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}}\phi_{i,Z} &= \phi_{i,N\setminus i} + \sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}}\phi_{i,Z}\\
&= -x_{0i} + \sum_{j\in N\setminus i} x_{ij}^{N\setminus\{i,j\}} + \sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}}\left(-\sum_{j\in N\setminus(Z\cup i)} x_{ji}^{Z} + \sum_{j\in Z} x_{ij}^{Z\setminus j}\right)\\
&= -x_{0i} + \left(\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}}\sum_{j\in Z} x_{ij}^{Z\setminus j}\right) - \left(\sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}}\sum_{j\in N\setminus(Z\cup i)} x_{ji}^{Z}\right)
\end{aligned}$$

We compare this to (2.17):

$$\lambda_{i,W}:\; -x_{0i} + \left(\sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ U\cup j\supseteq W}} x_{ij}^{U}\right) - \left(\sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ U\supseteq W}} x_{ji}^{U}\right) = 0$$

We will demonstrate that these summations are in fact the same by using bijective arguments. First, we will compare the positive terms:

$$\text{From } \phi_{i,Z}: \qquad \left(\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}}\sum_{j\in Z} x_{ij}^{Z\setminus j}\right), \tag{2.33}$$

$$\text{From } \lambda_{i,W}: \qquad \left(\sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ U\cup j\supseteq W}} x_{ij}^{U}\right) \tag{2.34}$$

We will rewrite the second summation (2.34). Let $Z = U\cup j$:

$$\sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ U\cup j\supseteq W}} x_{ij}^{U} = \sum_{j\in N\setminus i}\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W\\ j\in Z}} x_{ij}^{Z\setminus j} = \sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}}\sum_{j\in Z} x_{ij}^{Z\setminus j} \tag{2.35}$$

Thus we see that (2.33) and (2.34) are the same.

We now consider the negative terms:

$$\text{From } \phi_{i,Z}: \qquad -\left(\sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}}\sum_{j\in N\setminus(Z\cup i)} x_{ji}^{Z}\right) \tag{2.36}$$

$$\text{From } \lambda_{i,W}: \qquad -\left(\sum_{j\in N\setminus i}\sum_{\substack{U\subseteq N\setminus\{i,j\}\\ U\supseteq W}} x_{ji}^{U}\right) \tag{2.37}$$

Each term in either summation appears at most once. We show that these two summations are the same through the following:

Observation 1. *For any $Z \supseteq W$, $j \in N$, the following holds:*

$$\begin{gathered} Z \subsetneq N\setminus i, \textit{ and } j \in N\setminus(Z\cup i) \\ \Longleftrightarrow \\ j\in N\setminus i, Z\subseteq N\setminus\{i,j\} \end{gathered}$$

Using this observation, and by letting $Z = U$, we see that the sums (2.36) and (2.37) are equal. Thus, we have shown that the constraints are equal.

We will use a similar technique for the constraint associated with $\mu_{i,W}$. We define $\eta_{i,Z}$, for $Z \subsetneq N\setminus i$, as the following:

$$\eta_{i,Z} = \sum_{j\in N\setminus(Z\cup i)} x_{ij}^{N\setminus Z\cup\{i,j\}} - \sum_{j\in Z} x_{ji}^{N\setminus(Z\cup i)}$$

Again, for notational reasons we specifically define $\eta_{i,N\setminus i}$ as follows:

$$\eta_{i,N\setminus i} = x_{i0} - \sum_{j\in N\setminus i} x_{ji}^{\emptyset}$$

From here, we write out $\sum_{Z\subseteq N\setminus i, Z\supseteq W}\eta_{i,Z}$

$$\begin{aligned}
\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}} \eta_{i,Z} &= \eta_{i,N\setminus i} + \sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}} \eta_{i,Z}\\
&= x_{i0} - \sum_{j\in N\setminus i} x_{ji}^{\emptyset} + \sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}} \left(\sum_{j\in N\setminus (Z\cup i)} x_{ij}^{N\setminus(Z\cup\{i,j\})} - \sum_{j\in Z} x_{ji}^{N\setminus(Z\cup i)} \right)\\
&= x_{i0} - \left(\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}} \sum_{j\in Z} x_{ji}^{N\setminus(Z\cup i)} \right) + \left(\sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}} \sum_{j\in N\setminus(Z\cup i)} x_{ij}^{N\setminus Z\cup\{i,j\}} \right)
\end{aligned}$$

We compare this to (2.18):

$$\mu_{i,W} : x_{i0} - \left(\sum_{j\in N\setminus i} \sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup i)\supseteq W}} x_{ji}^{U} \right) + \left(\sum_{j\in N\setminus i} \sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup\{i,j\})\supseteq W}} x_{ij}^{U} \right) = 0$$

Again, we will demonstrate that these summations are in fact the same by using bijective arguments. First, we will compare the positive terms:

$$\text{From } \eta_{i,Z}: \quad \left(\sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}} \sum_{j\in N\setminus(Z\cup i)} x_{ij}^{N\setminus Z\cup\{i,j\}} \right), \tag{2.38}$$

$$\text{From } \mu_{i,W}: \quad \left(\sum_{j\in N\setminus i} \sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup\{i,j\})\supseteq W}} x_{ij}^{U} \right) \tag{2.39}$$

We will rewrite the second summation (2.39). Let $Z = N \setminus (U \cup \{i, j\})$ (and thus $U = N \setminus (Z \cup \{i, j\})$):

$$\sum_{j\in N\setminus i} \sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup\{i,j\})\supseteq W}} x_{ij}^{U} = \sum_{j\in N\setminus i} \sum_{\substack{Z\subseteq N\setminus\{i,j\}\\ Z\supseteq W}} x_{ij}^{N\setminus(Z\cup\{i,j\}} = \sum_{\substack{Z\subsetneq N\setminus i\\ Z\supseteq W}} \sum_{j\in N\setminus(Z\cup i)} x_{ij}^{N\setminus Z\cup\{i,j\}}$$

where the last equality holds because of Observation 1.

We now consider the negative terms:

$$\text{From } \eta_{i,Z}: \qquad -\left(\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}} \sum_{j\in Z} x_{ji}^{N\setminus(Z\cup i)}\right) \tag{2.40}$$

$$\text{From } \mu_{i,W}: \qquad -\left(\sum_{j\in N\setminus i} \sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup i)\supseteq W}} x_{ji}^{U}\right) \tag{2.41}$$

We rewrite (2.41) by letting $Z = N \setminus (U \cup i)$. Since $j \notin U$, this implies that $j \in Z$, and that $Z \subseteq N \setminus i$.

$$-\sum_{j\in N\setminus i} \sum_{\substack{U\subseteq N\setminus\{i,j\}\\ N\setminus(U\cup i)\supseteq W}} x_{ji}^{U} = -\sum_{j\in N\setminus i} \sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W\\ j\in Z}} x_{ji}^{N\setminus(Z\cup i)} = -\sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}} \sum_{j\in Z} x_{ji}^{N\setminus(Z\cup i)}$$

as required.

In conclusion, we have shown that the constraints associated with $\lambda_{i,W}$ and $\mu_{i,W}$ can be decomposed into a sum of terms based on the supersets Z of W, $\phi_{i,Z}$ and $\eta_{i,Z}$ respectively. In other words, the constraints can be rewritten as follows:

$$\lambda_{i,W} \quad : \qquad \sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}} \phi_{i,Z} = 0$$

$$\mu_{i,W} \quad : \qquad \sum_{\substack{Z\subseteq N\setminus i\\ Z\supseteq W}} \eta_{i,Z} = 0$$

□

In the next section, we will show that each of the ϕ and η terms are equal to 0 through our variable setting, and thus the constraints are satisfied.

Proof of Lemma 3. We have shown in Lemma 2 that for a city i and set of cities $W \subseteq N \setminus i$ such that $|W| \geq n - t$, the constraint associated with $\lambda_{i,W}$ in the primal is the sum of $\phi_{i,Z}$ terms. We will now prove that each $\phi_{i,Z}$ term is equal to 0 from the way that we've set our variables, and thus the whole constraint is satisfied.

Recall the definition of $\phi_{i,Z}$, and combine it with our variable setting (2.28), (2.29), and (2.30) when given a solution $(\hat{x}, \hat{\lambda})$ to BCP_{t+1}. Since $Z \supseteq W$, and $|W| \geq n - t$, then $|Z| \geq n - t$. Thus, we can use our definition of Q_Z^{t+1}:

$$\begin{aligned}
\phi_{i,Z} &= - \sum_{j \in N \setminus (Z \cup i)} x_{ji}^Z + \sum_{k \in Z} x_{ik}^{Z \setminus k} \\
&= - \left(\sum_{j \in N \setminus (Z \cup i)} \sum_{q \in Q_Z^{t+1}} d_{ji}^{q,|N \setminus Z|} \cdot \hat{\lambda}_q \right) + \left(\sum_{k \in Z} \sum_{q \in Q_{Z \setminus k}^{t+1}} d_{ik}^{q,|N \setminus (Z \setminus k)|} \cdot \hat{\lambda}_q \right)
\end{aligned}$$

First, we note that $|N \setminus (Z \setminus k)| = |N \setminus Z| + 1$. To proceed, it would be convenient to factor out the $\hat{\lambda}_q$ terms, but the two sums are indexed by different sets of q-routes, Q_Z^{t+1} and $Q_{Z \setminus k}^{t+1}$. However, the following claim shows that we can rewrite the second sum in a more convenient way:

Claim 1.

$$\sum_{k \in Z} \sum_{q \in Q_{Z \setminus k}^{t+1}} d_{ik}^{q,|N \setminus Z|+1} \cdot \hat{\lambda}_q = \sum_{k \in Z} \sum_{q \in Q_Z^{t+1}} d_{ik}^{q,|N \setminus Z|+1} \cdot \hat{\lambda}_q$$

Proof. We will prove our claim by showing that any q-route in $q \in Q_{Z \setminus k}^{t+1}$ for which $d_{ik}^{q,|N \setminus Z|+1} = 1$ is also in Q_Z^{t+1}, and that the reverse is also true.

First, for any $k \in Z$, consider a q-route $q \in Q_{Z \setminus k}^{t+1}$ for which $d_{ik}^{q,|N \setminus Z|+1} = 1$. By definition, $\{v_q(1), ..., v_q(|N \setminus Z| + 1)\} = N \setminus (Z \setminus k) = (N \setminus Z) \cup \{k\}$. Since $d_{ik}^{q,|N \setminus Z|+1} = 1$, $v_q(|N \setminus Z|) = i$ and $v_q(|N \setminus Z| + 1) = k$. Recall that $|Z| \geq n - t$, which implies that $|N \setminus Z| + 1 \leq t + 1$. Since $q \in Q_{Z \setminus k}^{t+1} \subseteq Q^{t+1}$, q is $(t + 1)$-cycle-free, and therefore k could not have already been visited in the first $|N \setminus Z|$ cities after the depot. Therefore, $\{v_q(1), ..., v_q(|N \setminus Z|)\} = N \setminus Z$, and so $q \in Q_Z^{t+1}$.

Next, for any $k \in Z$, consider a q-route $q \in Q_Z^{t+1}$ for which $d_{ik}^{q,|N \setminus Z|+1} = 1$. By definition, $\{v_q(1), ..., v_q(|N \setminus Z|)\} = N \setminus Z$. Since $d_{ik}^{q,|N \setminus Z|+1} = 1$, $v_q(|N \setminus Z| + 1) = k$, and thus $\{v_q(1), ..., v_q(|N \setminus Z| + 1)\} = (N \setminus Z) \cup \{k\} = N \setminus (Z \setminus k)$. Therefore, q is also in $Q_{Z \setminus k}^{t+1}$. □

Because of this observation, we can replace $Q_{Z \setminus k}^{t+1}$ in the second summation with Q_Z^{t+1}, switch the order of the summands, and then factor out λ_q. Thus,

$$\begin{aligned}
\phi_{i,Z} &= -\left(\sum_{q\in Q_Z^{t+1}} \sum_{j\in N\setminus(Z\cup i)} d_{ji}^{q,|N\setminus Z|} \cdot \hat{\lambda}_q\right) + \left(\sum_{q\in Q_Z^{t+1}} \sum_{k\in Z} d_{ik}^{q,|N\setminus Z|+1} \cdot \hat{\lambda}_q\right) \\
&= \sum_{q\in Q_Z^{t+1}} \hat{\lambda}_q \cdot \left(- \sum_{j\in N\setminus(Z\cup i)} d_{ji}^{q,|N\setminus Z|} + \sum_{k\in Z} d_{ik}^{q,|N\setminus Z|+1}\right)
\end{aligned}$$

We now claim that this is equal to 0 because the value within the parentheses is 0:

Claim 2.

$$- \sum_{j\in N\setminus(Z\cup i)} d_{ji}^{q,|N\setminus Z|} + \sum_{k\in Z} d_{ik}^{q,|N\setminus Z|+1} = 0$$

for $Z \subseteq N \setminus i$ such that $|Z| \geq n-t$, for any q-route $q \in Q_Z^{t+1}$.

Proof. Consider any route $q \in Q_Z^{t+1}$. By definition of d, all terms are 0 if $v_q(|N \setminus Z|) \neq i$. If $v_q(|N \setminus Z|) = i$, then since $q \in Q_Z^{t+1}$, $v_q(|N \setminus Z| - 1) \in (N \setminus Z) \setminus \{i\} = N \setminus (Z \cup i)$. Thus, there is exactly one $j \in N \setminus (Z \cup i)$ so that $d_{ji}^{q,|N\setminus Z|} = 1$, and therefore the left sum is -1. Now consider the right sum. Since $|N \setminus Z| + 1 \leq t+1$, $v_q(|N \setminus Z| + 1) \notin N \setminus Z$, since q is $(t+1)$-cycle-free. Thus, $v_q(|N \setminus Z| + 1) = k$ for exactly one $k \in Z$, and so the right sum is 1. These two sums combine to 0, as claimed. □

We conclude that, given i and W such that $W \subseteq N \setminus i$, $|W| \geq n-t$, $\phi_{i,Z}$ is zero for all $Z \subseteq N \setminus i$, $Z \supseteq W$. Therefore, the constraint associated with $\lambda_{i,W}$ is satisfied.

We use a similar argument to show that $\eta_{i,Z} = 0$ for every $Z \subseteq N \setminus i$, $Z \supseteq W$. Recall the definition of $\eta_{i,Z}$:

$$\eta_{i,Z} = \sum_{k\in N\setminus(Z\cup i)} x_{ik}^{N\setminus(Z\cup\{i,k\})} - \sum_{j\in Z} x_{ji}^{N\setminus(Z\cup i)}$$

For notational reasons, we will substitute $X = N \setminus (Z \cup i)$.

$$\eta_{i,Z} = \sum_{k\in X} x_{ik}^{X\setminus k} - \sum_{j\in N\setminus(X\cup i)} x_{ji}^{X}$$

Since $|Z| \geq n - t$, $|Z \cup i| \geq n - t + 1$, and thus $|X| = N \setminus (Z \cup i) \leq t - 1$, allowing us to use $\bar{Q}_X^{t+1}$. After variable replacement, and swapping the order of the sums, we get the following:

$$\eta_{i,Z} = -\left(\sum_{j \in N \setminus (X \cup i)} \sum_{q \in \bar{Q}_X^{t+1}} d_{ji}^{q,|N \setminus X|} \cdot \hat{\lambda}_q\right) + \left(\sum_{k \in X} \sum_{q \in \bar{Q}_{X \setminus k}^{t+1}} d_{ik}^{q,|N \setminus (X \setminus k)|} \cdot \hat{\lambda}_q\right)$$

Again, we note that $|N \setminus (X \setminus k)| = |N \setminus X| + 1$. As with the ϕ sums, we will be able to factor by $\hat{\lambda}_q$ because of the following claim:

Claim 3.

$$\sum_{k \in X} \sum_{q \in \bar{Q}_{X \setminus k}^{t+1}} d_{ik}^{q,|N \setminus X|+1} \cdot \hat{\lambda}_q = \sum_{k \in X} \sum_{q \in \bar{Q}_X^{t+1}} d_{ik}^{q,|N \setminus X|+1} \cdot \hat{\lambda}_q$$

Proof. As before, we will show that for a given $k \in X$, if $q \in \bar{Q}_{X \setminus k}^{t+1}$ such that $d_{ik}^{q,|N \setminus X|+1} = 1$, then $q \in \bar{Q}_X^{t+1}$, as well as the reverse.

For a given $k \in X$, consider the q-route $q \in \bar{Q}_{X \setminus k}^{t+1}$ such that $d_{ik}^{q,|N \setminus X|+1} = 1$. By definition, $v_q(|N \setminus X|) = i$ and $v_q(|N \setminus X| + 1) = k$. Since $q \in \bar{Q}_{X \setminus k}^{t+1}$, $\{v_q(|N \setminus X| + 2), ..., v_q(n)\} = X \setminus k$. Therefore, the set $\{v_q(|N \setminus X| + 1), ..., v_q(N)\} = (X \setminus k) \cup \{k\} = X$, and thus $q \in \bar{Q}_X^{t+1}$.

Now consider a q-route $q \in \bar{Q}_X^{t+1}$ such that $d_{ik}^{q,|N \setminus X|+1} = 1$. Since $q \in \bar{Q}_X^{t+1}$, and $v_q(|N \setminus X| + 1) = k$, it must be the case that $\{v_q(|N \setminus X| + 2), ..., v_q(n)\} = X \setminus k$. Therefore, $q \in \bar{Q}_{X \setminus k}^{t+1}$. □

We now replace $\bar{Q}_{X \setminus k}^{t+1}$ with $\bar{Q}_X^{t+1}$, swap the summand order, and factor:

$$\eta_{i,Z} = -\left(\sum_{q \in \bar{Q}_X^{t+1}} \sum_{j \in N \setminus (X \cup i)} d_{ji}^{q,|N \setminus X|} \cdot \hat{\lambda}_q\right) + \left(\sum_{q \in \bar{Q}_X^{t+1}} \sum_{k \in X} d_{ik}^{q,|N \setminus X|+1} \cdot \hat{\lambda}_q\right) \tag{2.42}$$

$$= \sum_{q \in \bar{Q}_X^{t+1}} \hat{\lambda}_q \cdot \left(-\sum_{j \in N \setminus (X \cup i)} d_{ji}^{q,|N \setminus X|} + \sum_{k \in X} d_{ik}^{q,|N \setminus X|+1}\right) \tag{2.43}$$

As before, we will show the quantity within the parentheses is always 0.

Claim 4.

$$-\sum_{j\in N\setminus(X\cup i)} d_{ji}^{q,|N\setminus X|} + \sum_{k\in X} d_{ik}^{q,|N\setminus X|+1} = 0 \tag{2.44}$$

for any q-route $q \in \bar{Q}_X^{t+1}$.

Proof. If $v_q(|N \setminus X|) \neq i$, then all terms are 0. Consider the case when $v_q(|N \setminus X|) = i$. Since $q \in \bar{Q}_X^{t+1}$, $\{v_q(|N \setminus X| + 1), ..., v_q(n)\} = X$. If $v_q(|N \setminus X| - 1) \in X \cup i$, it would violate the fact that q is $(t + 1)$-cycle-free, since $|X \cup i| \leq t$. Thus, there is exactly one $j \in N \setminus (X \cup i)$ such that $v_q(|N \setminus X| - 1) = j$. By definition of $\bar{Q}_X^{t+1}$, we know that $v_q(|N \setminus X| + 1) \in X$. Thus, these two sums combine to 0. □

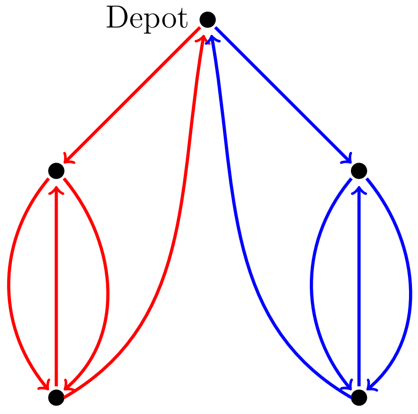
The proofs for $\phi_{i,N\setminus i}$ and $\eta_{i,N\setminus i}$ are similar to the other cases, with some minor notational differences. Thus, we have shown that $\phi_{i,Z} = 0$ and $\eta_{i,Z} = 0$ for all $Z \subseteq N\setminus i$, $|Z| \geq n-t$. □

2.5 The need for Subtour Elimination constraints

In the previous section, we showed that the BCP_{t+1} relaxation is at least as good as the ALP_t relaxation. The BCP_{t+1} relaxation combines column generation with q-route variables, and cut generation with subtour elimination constraints. One question that can be asked is where the strength of the BCP_{t+1} relaxation comes from, or, in other words, is it possible that the BCP_{t+1} relaxation is stronger than the ALP_t relaxation even if we drop the subtour elimination constraints? Dropping the subtour elimination constraints leads to a formulation that we call the q-route formulation. In this section, we give an example that answers this question for $t = 0$.

Recall that [24] showed that the optimal solution for ALP_0 is equivalent to the Held-Karp bound. The graph in Figure 2.1 demonstrates a solution to the q-route formulation that is not feasible for the Held-Karp formulation. The red walk and the blue walk are each q-routes of weight $\frac{1}{2}$, ensuring that our in-degree and out-degree constraints are satisfied. We can see, however, that subtour elimination constraints would not be satisfied by this example.

Claim 5. *Our example demonstrates that the HK bound can be strictly better than the q-route bound.*



The red and blue q-routes have weight $\frac{1}{2}$.

Figure 2.1: BCP vs HK

Proof. For the graph given in Figure 2.1, assume all costs going in and out of the 2 rightmost vertices are M, a non-negative constant, and all other costs equal to 0. This can be achieved even in Euclidean embeddings, by putting the two leftmost vertices in the same location as the depot and the 2 rightmost vertices at the same location at a distance of M from the root.

Formally, let S be the 2 rightmost vertices. We have that $c_a = M$ if $a \in \delta^-(S) \cup \delta^+(S)$, and $c_a = 0$ otherwise. Let each q-route have weight $\frac{1}{2}$. Then the cost of the red q-route is 0, and the cost of the blue q-route is $\frac{1}{2}(M + M) = M$.

Now, let $\hat{x}$ be an optimal solution to this graph using the Held-Karp formulation. Any solution that is feasible for the Held-Karp formulation must satisfy the subtour elimination constraints. In particular, $\hat{x}(\delta^-(S)) \geq 1$ and $\hat{x}(\delta^+(S)) \geq 1$, where S is as we described previously. Since $\delta^+(S) \cap \delta^-(S) = \emptyset$, we have that

$$\begin{aligned}\sum_{a \in A} c_a \hat{x}_a &\geq \sum_{a \in \delta^-(S)} c_a \hat{x}_a + \sum_{a \in \delta^+(S)} c_a \hat{x}_a \\ &= M \left(\sum_{a \in \delta^-(S)} \hat{x}_a + \sum_{a \in \delta^+(S)} \hat{x}_a \right) \\ &\geq 2M\end{aligned}$$

This shows that the HK bound can be strictly better than the BCP_1 bound without subtour elimination constraints. □

2.6 Conclusion

We have shown that for any choice of t, the lower bound on TSP achieved by BCP using $(t+1)$-cycle-free q-routes is at least as tight as that achieved by the ALP lower bound. We have also shown that the strength of such a bound cannot be obtained by using each component (q-routes and subtour elimination constraints) of the BCP_{t+1} formulation separately. Something that remains to be investigated is if the two formulations ALP_t and BCP_{t+1} are in fact equivalent, or if BCP_{t+1} attains a strict dominance.

Chapter 3

Cranial Vault Remodeling

3.1 Introduction

Craniosynostosis is a condition that afflicts 1 in every 2000 newborn infants [23]. The soft bones of the skull prematurely fuse, resulting in visible facial deformities such as asymmetry or centered bulging. Left untreated, there are very serious risks for the infant, such as visual impairment and stunted mental development [14]. To correct this condition, patients undergo a form of surgery called Cranial Vault Remodeling, which is performed on 2 - 3 patients every month at The Hospital for Sick Children (SickKids). The surgeon removes a strip of bone above the eyebrows called the front orbital bar, and cuts it in several places, reshaping it into a suitable curvature.

Current methodologies require the artistic judgment of the performing surgeon [6]. Recently, SickKids has developed a system of generating 'ideal' skull curves for patients on an individual basis, based on previous patient history and pre-surgery CT scans. Using this ideal curve, a stainless steel template called a 'bandeau' is machined to assist in the surgery. The craniofacial surgeon performs incisions on the removed skull piece, a process called osteotomizing, and then presses the bone against the metal template and screws it in place.

The placement of these incisions on the strip of bone is still a subjective process. We are interested in determining a set of cut locations using mathematical methods in order to provide a consistent solution that ensures the cut skull is as similar to the ideal curvature as possible. This solution would be presented to the craniofacial surgeon as a guideline.

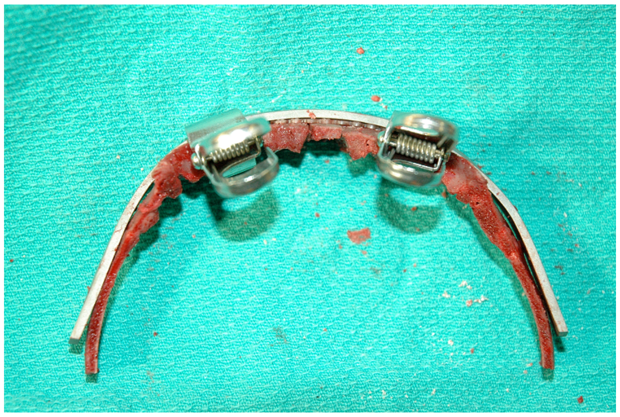


Figure 3.1: Orbital Bar clamped to Bandeau Template

3.2 Notation

In practice, the soft bones of the infant's skull are malleable and can be bent. However, one of our goals will be to reduce the amount of bending required to achieve the ideal shape. Therefore, we view the bone as a rigid material that will not bend, with cuts on the skull serving as hinges, allowing the cut segments to be repositioned relative to each other via rotation.

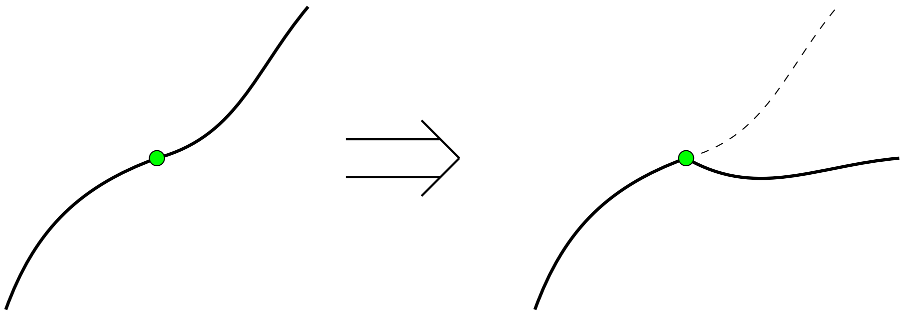


Figure 3.2: Rigid pieces rotated by Cut

We consider the malformed skull as a curve $f : [0,1] \to \mathbb{R}^2$, a continuous curve that is non-intersecting, i.e. $f(k) \neq f(k'),\ \forall\ k \neq k',\ k \in [0,1]$. We similarly let the ideal skull be represented by curve $g : [0,1] \to \mathbb{R}^2$. In practice, f and g are piecewise linear. The malformed skull is designated a midpoint, and a cut is made at this midpoint. An equal number of cuts is to be made on either side of this point, implying an odd number of cuts.

In practice, the total number of cuts is between 5 and 11 (2 to 5 on each side, with one in the middle).

We decouple the problem by considering only the right side of the malformed skull; our solution for determining the cut locations on the left is independent from the locations on the right, so the discussion applies for the left side as well. From this point on, we assume f describes the curve of the right side of the skull, with $f(0)$ being the midpoint of the entire skull, i.e. the left endpoint of the right curve. Similarly, g describes the curve of the right side of the ideal skull.

The number of cuts, T, is a fixed parameter, and we wish to determine the locations of these cuts on f, $C = \{c_0, c_1, ..., c_T\} \in [0,1]$, with $c_0 = 0$ and $c_T = 1$, serving as the endpoints of the right side. Without loss of generality, we assume that $c_0 < c_1 < ... < c_T$. We chose to solve the problem for fixed T, and then compare the optimal solutions for different values of T. This not only simplifies the optimization, but also gives the surgeon more options to choose from. Generally, we have seen that more cuts results in a better solution, but evaluating if the improved result is worth the additional cuts is difficult to model, and therefore left for the surgeon's judgement.

In practice, after a segment of the skull is cut, the endpoints are 'clamped' to locations on the metal bandeau template using screws. Thus, we introduce the notion of 'mapping'; a cut applied to the abnormal skull will correspond with a unique location on the ideal skull, where the endpoint of the skull segment defined by the cuts are clamped to. Given a set of cuts C, let $\phi_C : \{c_0, ..., c_T\} \to [0,1]$ be the function defining these mappings; if $\phi_C(c_i) = x$, then we say that cut i at location $f(c_i)$ maps to $g(x)$ on the ideal curve. The formal definition of this function will be presented later.

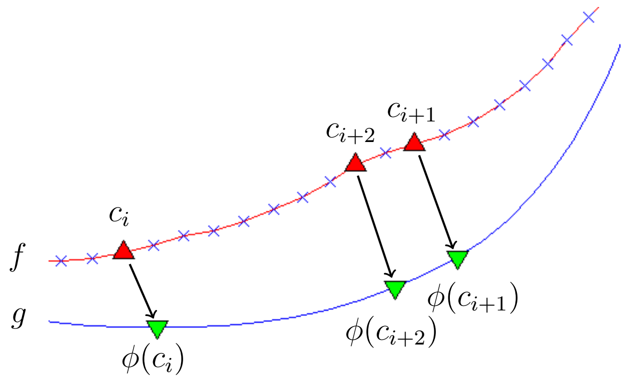


Figure 3.3: The ϕ_C function

The first cut, c_0, is fixed to be the midpoint of the malformed skull, i.e. $c_0 = 0$. Additionally, it is mapped to the midpoint of the ideal skull, so we set $\phi_C(c_0) = 0$. The last cut c_T is also fixed to be the right endpoint of the malformed skull, i.e. $c_T = 1$. We do not, however, fix $\phi_C(c_T)$ for feasibility reasons.

Given two points $a, b \in [0, 1]$ corresponding to two points on the curve f, and an $x \in [0, 1]$ corresponding to a point on g, we define $\sigma(a, b, x)$ to be the following:

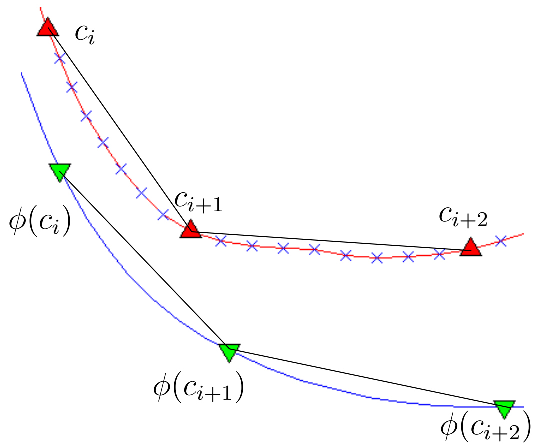
$$\sigma(a, b, x) = \inf\{y \in [0, 1] : y \geq x, ||g(y) - g(x)||_2 = ||f(a) - f(b)||_2\}$$

In other words, y corresponds to the next point on g after $g(x)$ such that the Euclidean distance between $g(x)$ and $g(y)$, the length of the 'chord' between them, is the same as the Euclidean distance between $f(a)$ and $f(b)$. Since we are minimizing a continuous function over a compact set, we may assume that the infimum is in fact a minimum. If such a y does not exist (i.e. the chord of $f(a)$ and $f(b)$ is too long for the curve g after $g(x)$), then we consider the cut that would require such a mapping as infeasible.

With this function, we now define the rest of $\phi_C(c_i)$ recursively, for all cuts $i > 0$.

$$\phi_C(c_i) = \sigma(c_{i-1}, c_i, \phi_C(c_{i-1})) \qquad \forall\ i > 0$$

From this recursive definition, we see that the location on the ideal skull that cut i at $f(c_i)$ maps to depends on the location of cut $i - 1$ and the location on the ideal skull that it has been mapped to.



The chords have the same length

Figure 3.4: Recursive Definition of ϕ

3.3 Calculating Costs

Any two adjacent cuts, c_i and c_{i+1}, define a segment of the curve f which we will call f_i, i.e. $f_i(x) = f(x)$ for $x \in [c_i, c_{i+1}]$. Their corresponding points on g similarly define a segment of the ideal skull, i.e. $g_i(y) = g(y)$ for $y \in [\phi_C(c_i), \phi_C(c_{i+1})]$. By definition of ϕ_C, the straightline distance between the endpoint of these two curves are equal, i.e.

$$||f_i(c_i) - f_i(c_{i+1})||_2 = ||g_i(\phi_C(c_i) - g_i(\phi_C(c_{i+1}))||_2$$

We will rotate and shift this segment of f in order to 'align' it with its corresponding segment of g. Define θ_{f_i} as the angle made by the endpoints of $f_i(x)$ and the x axis, i.e.

$$\theta_{f_i} = \arctan\left(\frac{f_i(c_{i+1})_y - f_i(c_i)_y}{f_i(c_{i+1})_x - f_i(c_i)_x}\right)$$

θ_{g_i} is defined similarly.

Thus, to align the endpoints of f_i and g_i for a given pair of adjacent cuts, we will rotate f_i by $\theta_i := \theta_{g_i} - \theta_{f_i}$ and translate the curve. We define the curve $\hat{f}_i$ as the following:

$$\hat{f}_i(x) = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix} \cdot f_i(x) + [g_i(\phi_C(c_i)) - f_i(c_i)]$$

After this transformation, we see that $\hat{f}_i(c_i) = g_i(\phi_C(c_i))$, and $\hat{f}_i(c_{i+1}) = g_i(\phi_C(c_{i+1}))$, i.e. $\hat{f}_i$'s and g_i's endpoints as defined by their domains are in the same coordinates of $\mathbb{R}^2$.

Since $\hat{f}_i$ and g_i share the same endpoints in $\mathbb{R}^2$, they define a region. We now make another assumption on $\hat{f}_i$ and g_i. Let $x, y \in [c_i, c_{i+1}]$, and $y > x$. We assume that

$$(f(y) - f(x))^\intercal \cdot (f(c_{i+1}) - f(c_i)) \geq 0$$

i.e. the angle formed by the chord between any two points on $\hat{f}_i$ and the chord of the endpoints of $\hat{f}_i$ is always acute with respect to the positive direction. We make a similar assumption on the chords of g_i. Examples of acceptable and unacceptable curves are shown in Figure 3.5.

Because of these assumptions, if we consider the chord of $\hat{f}_i$ to be the x-axis, the curve of $\hat{f}_i$ behaves like a regular function of one variable under this reparameterization. Therefore, $\hat{f}_i$ and g_i can be written as functions on this set of axes, on the domain $x \in [a, b]$ for some $a, b \in \mathbb{R},\ a \leq b$. We note that these assumptions are made for ease of presentation, and are not crucial to our algorithm.

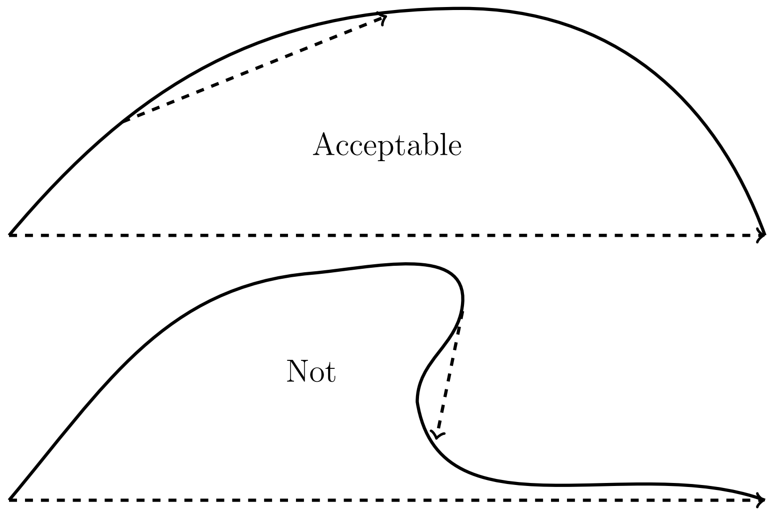


Figure 3.5: Curve assumptions

Given this reparameterization, we can now define the cost of mapping two segments of the skulls to simply be the absolute area difference between them via integration, i.e.

$$cost(\hat{f}_i, g_i) = \int_a^b |\hat{f}_i(x) - g_i(x)| \, \mathrm{d}x$$

Since $\hat{f}_i$ and g_i are defined by c_i, c_{i+1} and $\phi_C(c_i)$, $\phi_C(c_{i+1})$ respectively, we can redefine cost with those variables:

$$cost(c_i, c_{i+1}, \phi_C(c_i), \phi_C(c_{i+1})) = cost(\hat{f}_i, g_i)$$

The reparameterization is only required to simplify calculations and notation; in practice, since each segment is piecewise linear, the trapezoid method can be used to calculate their area difference.

3.4 Integer Programming Model

3.4.1 Formulation

With our notation definitions, we can describe an Integer Programming model. We first note that for our purposes, this was not the best approach computationally. However, the problem of determining the set of cut locations on our 2-dimensional curve is a subproblem of a more complex 3-dimensional problem. Our IP model may be easier to adapt for the 3-dimensional problem, and so we present it here.

We wish to determine the values for the set of cuts $C = \{c_0, ..., c_T\}$ in order to minimize the total cost of each adjacent pair of cuts. In other words, we wish to find C in order to minimize

$$cost(C) = \sum_{i=0}^{T-1} cost(c_i, c_{i+1}, \phi_C(c_i), \phi_C(c_{i+1}))$$

For our IP formulation, we chose to discretize the malformed skull f into a set of locations L, and similarly discretize the ideal skull g into a set of locations I. Let $L = \{\ell_0, \ell_1, ..., \ell_{n-1}\}$ be a predetermined set of n locations, such that $\ell_i \in [0, 1]$ for each i. Without loss of generality, let $\ell_0 < \ell_1 < ... < \ell_{n-1}$, with $\ell_0 = 0$ and $\ell_{n-1} = 1$. We then restrict our cut locations to be from L, i.e. $c_i \in L$ for each i. In practice, the set L is chosen so that each $f(l_i)$ is evenly spaced across the curve, though this is not required.

Similarly, the function ϕ_C will need to be redefined so that it maps from C to the set of discrete locations on the ideal skull, $I = \{q_0, q_1, ..., q_{m-1}\}$, for some choice of m. In practice, it is preferable that m and n are chosen so that $||f(\ell_i) - f(\ell_{i+1})||_2 \approx ||g(q_k) - g(q_{k+1})||_2$ for any choice of index $0 \leq i \leq n-1$ and $0 \leq k \leq m-1$, so that as many possible feasible mapping are considered.

The redefining of ϕ_C also requires the redefining of σ. Consider a pair of cuts c_i and $c_{i+1} \in L$ on the malformed skull, with $\phi_C(c_i) = q_k$ for some k. Then we define σ as

$$\sigma(c_i, c_{i+1}, q_k) = \min\{q_l \in I : l > k, |(||g(q_l) - g(q_k)||_2 - ||f(c_i) - f(c_{i+1})||_2)| \leq \epsilon\}$$

where ϵ is a pre-defined error tolerance parameter. This error tolerance is needed in order to obtain reasonable results in our model, due to the fact that both the ideal and malformed skulls are discretized. Otherwise, it would be unlikely that any $q_l \in I$ would satisfy $||g(q_l) - g(q_k)||_2 = ||f(c_i) - f(c_{i+1})||_2$ exactly.

Next, we introduce our decision variables. We define $y_{c,i}$ as follows

$$y_{c,i} = \begin{cases} 1 & : \text{if cut } c \text{ occurs at location } \ell_i \\ 0 & : \text{otherwise} \end{cases}$$

Due to the technical constraints of calculating costs in our model, it is necessary to define an additional variable $x^c_{i,j}$ as follows:

$$x^c_{i,j} = \begin{cases} 1 & : \text{if cut } c \text{ occurs at location } \ell_i \text{ and is mapped to location } q_j \\ 0 & : \text{otherwise} \end{cases}$$

Thus it follows that

$$y_{c,i} = \sum_j x^c_{i,j}$$

Let w_{i_1,i_2,j_1,j_2} be defined as *cost*(i_1, i_2, j_1, j_2), where *cost* is defined as in the previous section, where we use any cut index c. If the mapping of i_1 and i_2 to j_1 and j_2 is not suitable for our error tolerance, i.e.

$$|(||g(j_2) - g(j_1)||_2 - ||f(i_2) - f(i_1)||_2)| > \epsilon$$

then we define it as 0 and say that (i_1, i_2) is *incompatible* with (j_1, j_2), leaving such a mapping to be prevented by the constraints of our IP.

Our formulation is a binary pure integer quadratic program. Let M be the index of the middle cut, i.e. $M = \frac{T-1}{2}$, and r be the index of the middle location in L.

$$
\begin{aligned}
\min \quad & \sum_{\substack{(i_1,j_1)\,\in\\ [n-1]\times[m-1]}} \sum_{\substack{(i_2,j_2)\,\in\\ [n-1]\times[m-1]}} \sum_{c\in[T-1]} x^c_{i_1,j_1}\cdot x^{c+1}_{i_2,j_2}\cdot w_{i_1,i_2,j_1,j_2} \\
\text{s.t.} \quad & \sum_{c\in[T-1]} \sum_{j\in[m-1]} x^c_{i,j} \le 1 \qquad \forall\ i \in [n-1] \\
& \sum_{c\in[T-1]} \sum_{i\in[n-1]} x^c_{i,j} \le 1 \qquad \forall\ j \in [m-1] \\
& \sum_{i,j} x^c_{i,j} = 1 \qquad \forall\ c \in [T-1] \\
& y_{c,i} \le \sum_{k>i} y_{c+1,k} \qquad \forall\ c \in [T-1] \\
& x^M_{r,s} = 1 \\
& x^c_{i_1,j_1} + \sum_{\substack{i_2,j_2\\ \text{incompatible}}} x^{c+1}_{i_2,j_2} \le 1 \qquad \forall\ (i_1,j_1),\ \forall\ c\in[T-1] \\
& \sum_{\substack{i_1,j_1\\ \text{incompatible}}} x^c_{i_1,j_1} + x^{c+1}_{i_2,j_2} \le 1 \qquad \forall\ (i_2,j_2),\ \forall\ c\in[T-1] \\
& \sum_j x^0_{0,j} = 1 \\
& \sum_j x^{T+1}_{n-1,j} = 1
\end{aligned}
$$

3.4.2 Constraints

$$
\min \quad \sum_{\substack{(i_1,j_1)\,\in\\ [n-1]\times[m-1]}} \sum_{\substack{(i_2,j_2)\,\in\\ [n-1]\times[m-1]}} \sum_{c\in[T-1]} x^c_{i_1,j_1}\cdot x^{c+1}_{i_2,j_2}\cdot w_{i_1,i_2,j_1,j_2}
$$

As x is an indicator variable, the cost $w_{i1,i2,j1,j2}$ only contributes to the objective function if both $x^c_{i_1,j_1}$ and $x^{c+1}_{i_2,j_2}$ are 1, indicating that some cut c is mapped from i_1 to j_1, and cut $c+1$ is mapped from i_2 to j_2. If this mapping is physically infeasible, then our constraints prevent the two associated x variables from both being 1.

$$\begin{aligned} \sum_{c\in[T-1]}\sum_{j\in[m-1]} x_{i,j}^c &\le 1 \quad \forall\, i \in [n-1] \\ \sum_{c\in[T-1]}\sum_{i\in[n-1]} x_{i,j}^c &\le 1 \quad \forall\, j \in [m-1] \end{aligned}$$

The first constraint ensures that for each location on the abnormal skull, there is at most one cut at that location. The second constraint ensures the feasibility for the mapped locations on the ideal skull.

$$\sum_{i,j} x_{i,j}^c = 1 \quad \forall\, c \in [T-1]$$

Each cut c must be assigned to some location pair i and j.

$$y_{c,i} \le \sum_{k>i} y_{c+1,k} \quad \forall\, c \in [T-1]$$

This constraint ensures that cut order is preserved; if cut c is assigned to location i, then the next cut $c+1$ must be assigned to a location after i.

$$x_{r,s}^M = 1$$

As mentioned, the middle cut must map the midpoint of the abnormal skull to the midpoint of the ideal skull.

$$\begin{aligned} x_{i_1,j_1}^c + \sum_{\substack{i_2,j_2 \\ \text{incompatible}}} x_{i_2,j_2}^{c+1} &\le 1 \quad \forall\, (i_1,j_1),\ \forall\, c \in [T-1] \\ \sum_{\substack{i_1,j_1 \\ \text{incompatible}}} x_{i_1,j_1}^c + x_{i_2,j_2}^{c+1} &\le 1 \quad \forall\, (i_2,j_2),\ \forall\, c \in [T-1] \end{aligned}$$

The first constraint ensures that if cut c is made from i_1 to j_1, then every other cut pair that would result in an infeasible mapping cannot be made as the next cut. The second constraint has the same purpose and tightens the relaxation.

3.5 Dynamic Programming Model

3.5.1 Formulation

To use our DP method, it is again necessary to discretize the malformed skull f into a set of locations L. However, it will not be necessary to discretize the ideal skull, and there will be no need for an error tolerance ϵ. As before, we will find a minimizer $C \subseteq L$, with the assumption that with appropriate choice of L, our solution will be close enough to the minimizer of $C \subseteq [0, 1]$. Let $L = \{l_0, l_1, ..., l_{n-1}\}$ be a predetermined set of n locations, such that $l_i \in [0, 1]$ for each i. Again, without loss of generality, let $l_0 < l_1 < ... < l_{n-1}$, with $l_0 = 0$ and $l_{n-1} = 1$. We then restrict our cut locations to be from L, i.e. $c_i \in L$ for each i.

We define the function $\Delta(k, a)$ as the minimum cost of making $k + 1$ cuts, with the rightmost (k^{th}) cut at location $f(l_a)$ and with the leftmost cut 0 at $l_0 = 0$. Thus to determine the optimal cost for the right side of the skull, we wish to know the value $\Delta(T, n - 1)$.

Additionally, we define a function $p(k, a)$. Assume that cut k is at location l_a, and all preceding cuts (between cut 0 and k) are arranged as in the optimal solution defined by $\Delta(k, a)$. Then $p(k, a)$ is the location that cut k at location l_a maps to given this configuration, i.e. $p(k, a) = \phi(c_k)$ assuming all previous cuts are defined as in $\Delta(k, a)$. Since the leftmost cut c_0 maps the abnormal location $f(0)$ to the ideal location $g(0)$, $p(0, 0) = 0$.

In our dynamic programming formulation, we define $\Delta(k, a)$ recursively, and populate a table of values returned by $\Delta(k, a)$ and $p(k, a)$ in order to re-use them in subsequent function calls. As a base case, consider $\Delta(0, 0)$. Since there are no cuts preceding c_0, the cut only defines a point on the curve, and so $\Delta(0, 0) = 0$. All higher indexed cuts are defined recursively:

$$\Delta(k,a) = \begin{cases} 0 & : \text{ if } k = a = 0 \\ \infty & : \text{ if } a < k \\ \min\{\Delta(k-1,b) + \mathit{cost}(b, a, p(k-1,b), \sigma(b, a, p(k-1,b))) : \forall\ b < a\} \\ & : \text{ otherwise} \end{cases}$$

The first case is the base case. The second case is an infeasible cut location, since two cuts cannot share the same location, and the set of cuts is ordered.

The third case is recursive. Since we are investigating the case when cut k is at location a, we next try to determine its left cut neighbour, i.e. the location of cut $k-1$. We enumerate through all possible locations b for this cut, with $0 \leq b < a$. If we place cut $k-1$ at location b, we know (through recursion) the cheapest cost of doing so is $\Delta(k-1,b)$. In addition to placing these cuts, we must also add in the cost of mapping the segment between locations b and a to the ideal skull. When evaluating $\Delta(k-1,b)$, we know that cut $k-1$ at location b maps to location $p(k-1,b)$, and thus location a maps to $\sigma(b,a,p(k-1,b))$. Thus, the cost of this segment is $cost(b,a,p(k-1,b))$.

Enumerating through all valid candidates b, let b' be the location that minimizes our total cost, i.e.

$$b' = \underset{0\leq b<a}{\text{argmin}} \Delta(k-1,b) + cost(b,a,p(k-1,b),\sigma(b,a,p(k-1,b))) \tag{3.1}$$

We then set $p(k,a)$ to $\sigma(b',a,p(k-1,b'))$ and set $\Delta(k,a)$ to be the cost as defined by b'.

3.5.2 Correctness

Theorem 4. *The Dynamic Programming formulation gives the optimal cut configuration for $\Delta(k,a)$, $\forall\ k \in \{0,...,T\}$, $\forall\ a \in \{0,...,n-1\}$.*

Proof. We will prove correctness of our dynamic program using induction on k in $\Delta(k,a)$, $\forall\ a \in \{0,...,n-1\}$.

Base Case: $\Delta(1,a)$, for some $a \in \{0,...,n-1\}$. Since there are only two cuts, c_0 and c_1, there are no decisions to be made and the cost is simply $0 + cost(c_0,a,\phi(c_0),\sigma(c_0,a,\phi(c_0)))$. From the definition of $p(0,0)$, we see that this is equivalent to $\phi(c_0)$, so the cost is equal to $\Delta(1,a)$.

Induction Hypothesis: Assume $\Delta(k,a)$ is the optimal cost of making cut k at any $a \in \{0,...,n-1\}$, $\forall\ k \leq m$.

Induction Step: We wish to calculate $\Delta(k,a)$, $a \in \{0,...,n-1\}$, $k = m+1$. We claim that the optimal arrangement of these cuts is as configured by $\Delta(m,b')$, where b' is as defined in (3.1). Consider any other arrangement of the cuts $0,...,m+1$, with cut $m+1$ at location a and cut 0 at location 0, but with the other cuts not necessarily as defined by $\Delta(m,b')$. Let cut m be at location $\bar{b}$. Thus the cost of this cut and its preceding cuts is at least $\Delta(m,\bar{b}) + cost(\bar{b},a,p(m,\bar{b}),\sigma(\bar{b},a,p(m,\bar{b})))$, since by induction, $\Delta(m,\bar{b})$ is the lowest possible cost of placing cuts $0,...,m$ with m at location $\bar{b}$. But b' is the minimizer of (3.1), so this cost cannot be less than $\Delta(k,a)$, as required. □

3.5.3 Complexity

Theorem 5. *The worst case runtime of our Dynamic Programming formulation is $O(T \cdot n^3)$, where T is the number of cuts and n is the number of locations.*

Proof. To determine $\Delta(T, n-1)$, at worst we must populate a table of values of $\Delta(k, a)$, for all $k \in \{0, ..., T\}$ and for all $a \in \{0, ..., n-1\}$, i.e. $T \cdot n$ values. We will analyze the runtime for each pair (k, a) starting with lower cut indexes, i.e. $k = 1$ (since $k = 0$ is a base case for Δ).

Let $k = 1$, and $a > 0$. Since c_0 is fixed, we only have to calculate $cost(0, a, 0, \sigma(0, a, 0))$. This involves comparing a piecewise linear curve of at most n points with a piecewise linear curve, almost of at most n points. Since the curves are piecewise linear, we can calculate the area difference using the trapezoid method, and thus we require at most $O(n)$ operations to do so.

Now consider any higher k. To calculate $\Delta(k, a)$, we find the minimizer b' of $\Delta(k-1, b) + cost(b, a, p(k-1, b), \sigma(b, a, p(k-1, b)))$. Since we populated a table of Δ and p values for cut indices lower than k, retrieving those values has constant runtime. The cost of calculating *cost* is again $O(n)$. There are at most n potential candidates for b, so the runtime is $O(n^2)$.

Therefore, the runtime of evaluating each label (k, a) has complexity $O(n^2)$, and there are at most $T \cdot n$ labels, so the runtime is at most $O(T \cdot n^3)$. □

3.6 Computational Results

The following section contains computational results from two test cases given to us by the Hospital for Sick Children; the metopic case is an example of centerline bulging, and the unicoronal case is an example of frontal asymmetry. For each case, we were given different resolutions to represent the skulls; the deformed skull (and its ideal skull counterpart) would be a curve formed by joining either 51, 101, or 201 ordered points. For the deformed skull, these points serve as the n predetermined candidate cut locations L. For each of these cases, we ran our DP and IP formulations for 5, 7, 9, and 11 cuts. Figure 3.6 is a comparison of the results generated for the Unicoronal case with 201 candidate locations. Figure 3.7 is a comparison of the Metopic case with 5 cuts.

The abnormal skull f is presented as a red curve, and the ideal skull g is presented as a blue curve below it. The blue `X` marks on the red skull represent the set of candidate cut locations L. In each diagram, the red triangles represent cut locations c_i on the abnormal skull, and the green triangles represent where they map to on the ideal skull, $\phi(c_i)$. Each deformed skull also has a pair of dummy cuts at its endpoints. The appearance of the abnormal skull after it has been cut and rotated is interposed as a red curve over the blue ideal skull in order to demonstrate how well the curves match. The objective values are the area between the red and blue curves, measured in mm^2.

Table 3.1 is a full comparison of our computational results, including runtimes and objectives from our integer programming model.

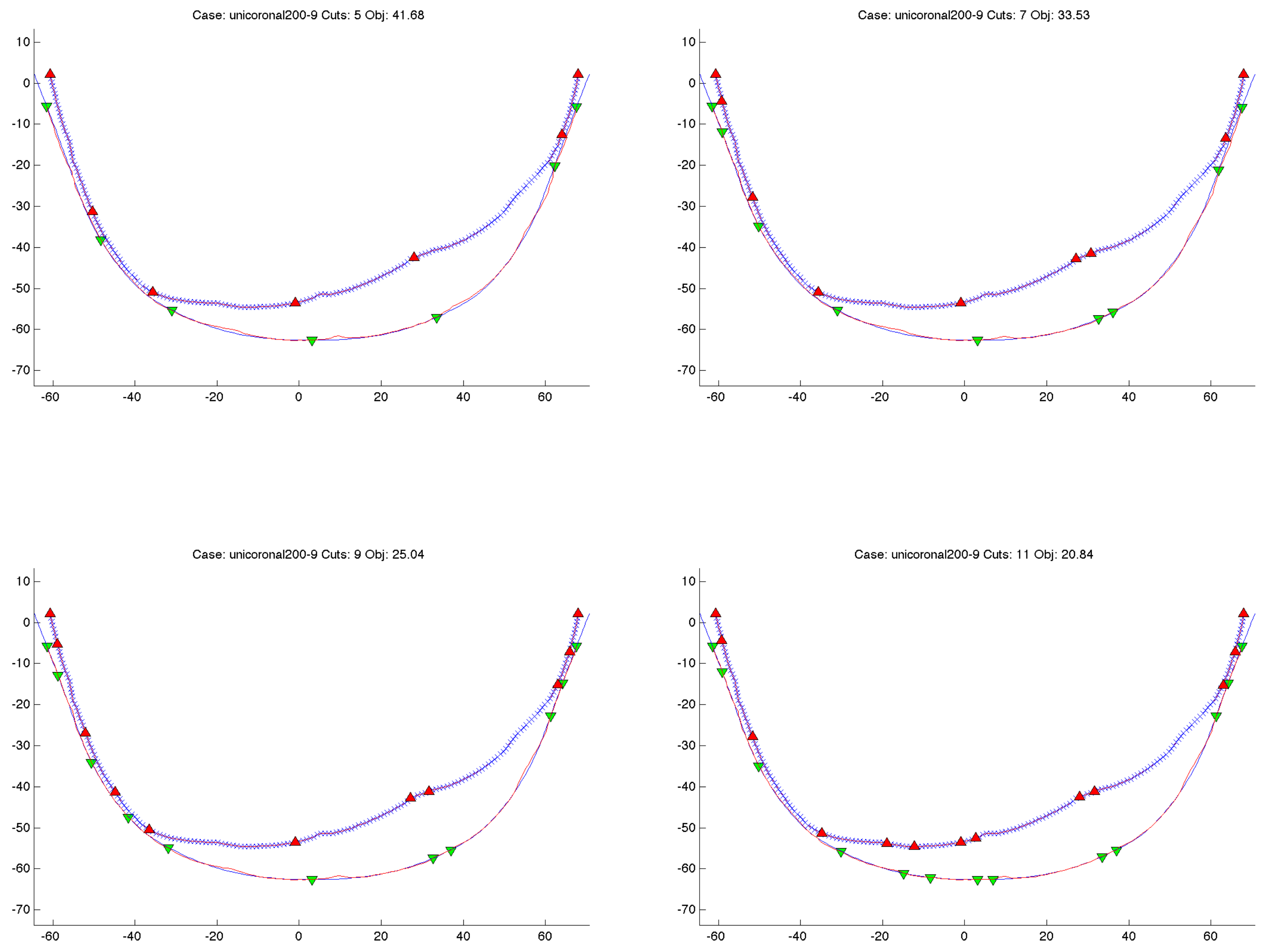


Figure 3.6: Unicoronal Comparison, 201 Cut Locations

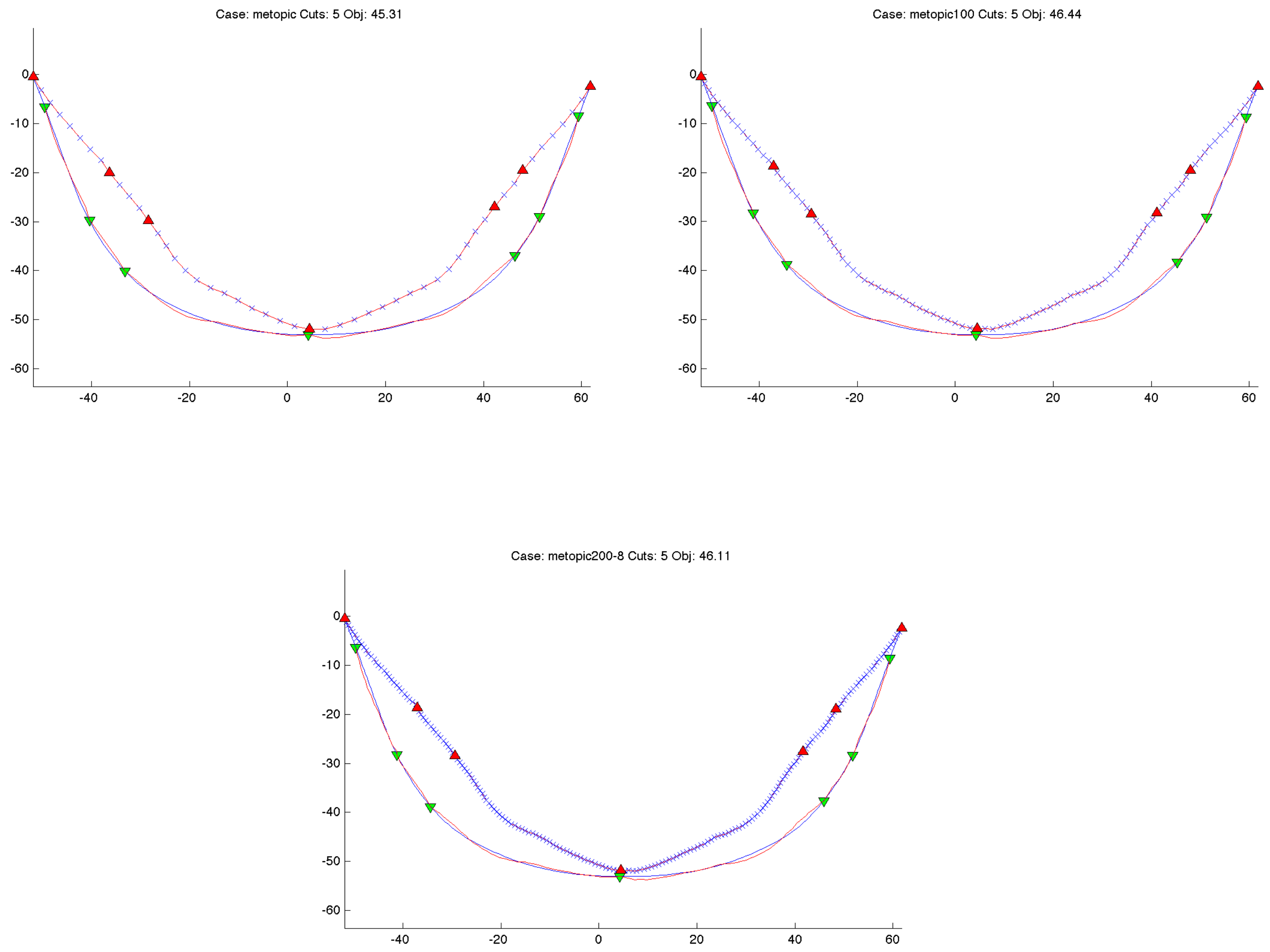


Figure 3.7: Metopic Comparison, 51, 101 and 201 Cut Locations with 5 cuts

Our tests were executed on a server with 48 AMD Opteron 6176 CPUs, running at 800 MHz each with 256 GB of RAM. Integer programs were solved using CPLEX 12.4, with parallelization enabled. Runtimes were limited to 1 hour.

Case Name	T	$\|L\|$	DP Runtime	DP Obj.	IP Runtime	IP Obj.
`metopic-5`	5	51	0.03s	45.31	30.31s	45.90
`metopic-7`	7	51	0.08s	35.96	40.06s	36.19
`metopic-9`	9	51	0.11s	28.91	50.49s	28.98
`metopic-11`	11	51	0.24s	24.89	1m0.7s	24.89
`metopic100-5`	5	101	0.11s	46.44	19m16s	49.83
`metopic100-7`	7	101	0.17s	36.32	22m32s	37.95
`metopic100-9`	9	101	0.29s	29.18	35m12s	31.92
`metopic100-11`	11	101	0.32s	22.89	37m11s	22.64
`metopic200-5`	5	201	0.88s	46.11	>1h	-
`metopic200-7`	7	201	1.64s	36.50	>1h	-
`metopic200-9`	9	201	2.37s	29.54	>1h	-
`metopic200-11`	11	201	2.98s	23.45	>1h	-
`unicoronal-5`	5	51	0.03s	43.43	30.22s	44.11
`unicoronal-7`	7	51	0.04s	32.48	40.27s	32.51
`unicoronal-9`	9	51	0.04s	25.02	49.25s	25.03
`unicoronal-11`	11	51	0.05s	20.87	1m0.4s	20.13
`unicoronal100-5`	5	101	0.10s	44.14	20m14s	43.44
`unicoronal100-7`	7	101	0.17s	34.64	23m10s	35.82
`unicoronal100-9`	9	101	0.22s	25.52	28m12s	26.93
`unicoronal100-11`	11	101	0.27s	22.42	35m28s	23.56
`unicoronal200-5`	5	201	0.92s	41.68	>1h	-
`unicoronal200-7`	7	201	1.59s	33.53	>1h	-
`unicoronal200-9`	9	201	2.30s	25.04	>1h	-
`unicoronal200-11`	11	201	2.96s	20.84	>1h	-

Table 3.1: Computational Results

Differences between the DP and IP objectives are due to the discretization of the ideal skull required for the IP model, as well as the error tolerance that was introduced. Objective values are measured in mm^2. We can see that the DP model is at least 2 orders of magnitude faster than the IP model, with the benefit of more accurate results.

Chapter 4

Future Work and Conclusions

In this thesis, we presented two applications of classic dynamic programming.

In Chapter 2, we presented BCP and ALP, two different families of lower bounds for the Traveling Salesman Problem. The Branch-Cut-and-Price lower bounds were obtained by using column generation and cut generation. Dynamic programming is the basis of the pricing algorithm used during the column generation. Our theoretical results prove that, given a solution to BCP_{t+1}, we can construct a feasible solution to ALP_t with the same objective value, demonstrating that the lower bound for BCP_{t+1} is at least as high as the lower bound for ALP_t, i.e. BCP_{t+1} dominates ALP_t. Future work could include determining if the dominance is strict, understanding the relationship between BCP and ALP, and designing new ways to approximate the shortest path LP used to solve the TSP. A possible direction would be to relate ALP with the ng-path relaxation presented by Baldacci *et al.* [2].

In Chapter 3, we presented our dynamic programming solution for determining the placement of cuts in Cranial Vault Remodeling. Our implementation of the model provided very fast results for the variety of cases and parameters given to us, providing a practical solution to surgeons who wish to use our model. However, our model is limited to 2-dimensions, essentially determining the placement of the strip of bone on the skull over the eyebrows (the frontal orbital bar). While the reshaping of the orbital bar is key to determining the placement of the rest of the bone pieces, the surgeons at SickKids have expressed interest in a 3-dimensional model to explicitly determine how the entire skull should be cut. Such a model would not be able to take advantage of the sequential cut structure used in the 2D model, thus limiting the usefulness of dynamic programming. The resulting pieces after the cuts are made could be rotated or moved to different parts of the skull. Additionally, the objective of such a model would be to minimize differences in volume, rather than area, which would present new challenges for calculating costs. A possible starting point would be to adapt our Integer Programming model, or to use our Dynamic Programming approach in multiple 2D ‘slices’ and combine the results into a 3D solution.

References

[1] E. Balas. Projection, lifting and extended formulation in integer and combinatorial optimization. *Ann. Oper. Res.*, 140:125–161, 2005.

[2] R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.*, 59(5):1269–1283, 2011.

[3] R. Bellman. Combinatorial processes and dynamic programming. In *Proc. Sympos. Appl. Math., Vol. 10*, pages 217–249. American Mathematical Society, Providence, R.I., 1960.

[4] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Mach.*, 9:61–63, 1962.

[5] R. Bellman. *Dynamic programming*. Princeton Landmarks in Mathematics. Princeton University Press, Princeton, NJ, 2010. Reprint of the 1957 edition.

[6] J. Burge, N. Saber, T. Looi, B. French, Z. Usmani, N. Anooshiravani, P. Kim, C. Forrest, and J. Phillips. Application of cad/cam prefabricated age-matched templates in cranio-orbital remodeling in craniosynostosis. *Journal of Craniofacial Surgery*, 22(5):1810–1813, Sep 2011.

[7] R. Carr and S. Vempala. On the Held-Karp relaxation for the asymmetric and symmetric traveling salesman problems. *Math. Program.*, 100(3, Ser. A):569–587, 2004.

[8] M. Charikar, M. Goemans, and H. Karloff. On the integrality ratio for the asymmetric traveling salesman problem. *Math. Oper. Res.*, 31(2):245–252, 2006.

[9] N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981.

[10] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, 1998. A Wiley-Interscience Publication.

[11] G. Dantzig, D. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *J. Operations Res. Soc. Amer.*, 2:393–410, 1954.

[12] L. Ford and D. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Sci.*, 5:97–101, 1958.

[13] R. Fukasawa, H. Longo, J. Lysgaard, M. de Aragão, M. Reis, E. Uchoa, and R. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Math. Program.*, 106(3, Ser. A):491–511, 2006.

[14] D. Gault, D. Renier, D. Marchac, and B. Jones. Intracranial pressure and intracranial volume in children with craniosynostosis. *Plastic and reconstructive surgery*, 90(3):377, 1992.

[15] M. Goemans. Worst-case comparison of valid inequalities for the TSP. *Math. Programming*, 69(2, Ser. A):335–349, 1995.

[16] M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Res.*, 18:1138–1162, 1970.

[17] S. Irnich and D. Villeneuve. The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS J. Comput.*, 18(3):391–406, 2006.

[18] N. Kaneshiro. Cranial sutures. *Medline Plus, U.S. National Library of Medicine*, 2011.

[19] J. Kleinberg and E. Tardos. *Algorithm Design*. Pearson Education, 1st edition, 2006.

[20] M. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, Nov - Dec 2005.

[21] M. Mangel and C. Clark. *Dynamic modeling in behavioral ecology*. Princeton University Press, 1989.

[22] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, Mar 1970.

[23] B. Slater, K. Lenton, M. Kwan, D. Gupta, D. Wan, and M. Longaker. Cranial sutures: a brief review. *Plastic and Reconstructive Surgery*, 121(4):–8, Apr 2008.

[24] A. Toriello. Equivalence of an approximate linear programming bound with the held-karp bound for the traveling salesman problem. *Submitted*, 2013.

[25] A. Toriello. Optimal toll design: A lower bound framework for the asymmetric traveling salesman problem. *Mathematical Programming A*, 2013.

[26] L. Wolsey. *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, 1998. A Wiley-Interscience Publication.

[27] M. Zuker. The use of dynamic programming algorithms in RNA secondary structure prediction. In *Mathematical methods for DNA sequences*, pages 159–184. CRC, Boca Raton, FL, 1989.