Title of Thesis with all Formula,
Symbols, or Greek Letters Written out in Words

by

Patrick Di Salvo

A Thesis

presented to

The University of Guelph

In partial fulfilment of requirements
for the degree of

Master of Pokemon, Poke master

in

Computer Science

Guelph, Ontario, Canada

ABSTRACT

Title of Thesis with all Formula, Symbols, or Greek Letters Written out in Words

Patrick Di Salvo
University of Guelph, 2099

Advisor:
Dr. Sherlock Holmes

Present your abstract here. This template is just an example of an outline to consider for your thesis. It is a good idea to double-check with the current University's thesis requirements here:

`https://www.uoguelph.ca/graduatestudies/current-students/preparation-your-thesis`

Acknowledgments

Present your acknowledgements here.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Amidakuji is a custom in Japan which allows for a pseudo-random assignment of children to prizes [**?**]. Usually done in Japanese schools, a teacher will draw N vertical lines, hereby known as lines, where N is the number of students in class. At the bottom of each line will be a unique prize. And at the top of each line will be the name of one of the students. The teacher will then draw 0 or more horizontal lines, hereby known as *bars*, connecting two adjacent lines. The more bars there are the more complicated (and fun) the Amidakuji is. No two endpoints of two bars can be touching. Each student then traces their line, and whenever they encounter an end point of a bar along their line, they must cross the bar and continue going down the adjacent line. The student continues tracing down the lines and crossing bars until they get to the end of the ladder lottery. The prize at the bottom of the ladder lottery is their prize [**?**]. See Fig. 1.2 for an example of a ladder lottery.

The word Amidakuji has an interesting etymology. In Japanese, Amida is the Japanese name for Amithaba, the supreme Buddha of the Western Paradise. See image —image ref— for a picture of Amithaba. Amithaba is a Buddha from India and there is a cult based around him. The cult of Amida, otherwise known as Amidism, believes that by worshiping Amithaba, they shall enter into the his Western Paradie.[**?**] Amidism began in India in the fourth century and made its way to China and Korea in the fifth century, and finally came to Japan in ninth century [**?**]. It was in Japan, where the game Amidakuji began. It is known as 'Ghost Legs' in China and Ladder Lotteries in English.

The game Amidakuji began in Japan in the Muromachi period, which spanned

from 1336 to 1573 [**?**]. During the Muromachi period, the game was played by having players draw their names at the top of the lines, and at the bottom of the lines were pieces of paper that had the amount the players were willing to bet. The pieces of paper were folded in the shape of Amithaba's halo, which is why the game is called Amidakuji. Kuji is the Japanese word for lottery. Hence the name of the game being Amidakuji.

1.1 Thesis Statement

This thesis provides four full, or partial, solutions to four problems related to ladder-lotteries. The first of these problems is the so called counting problem,, which asks, how many ladders are in $OptL\{\pi\}$? This thesis provides a formula for the exact number of ladders in $OptL\{\pi\}$, for certain cases of π as well as a general recurrence relation for $OptL\{\pi\}$ when π is the decending permutation. The second problem is the so called minimum height problem which asks, given all the ladders in $Optl\{\pi\}$, which ladder(s) are the shortest, that is to say which ladders have the smallest height? This thesis provides a theorem as to which ladder(s) in $OptL\{\pi\}$ have the shortest heights. The third problem is the so called canonical ladder generation problem. This problem asks, given all permutations of size N, is there a Gray Code to generate a canonical ladder from each permutation's $OptL\{\pi\}$? In other words, is there an easy way to transition from one permutation's canonical ladder to the next permutation's canonical ladder until all permutations of size N have had their canonical ladder generated. This thesis provides such a Gray Code. The last problem is the so called encoding problem. The encoding problem asks, given a ladder-lottery, provide an optimal compact representation of said ladder-lottery. This thesis provides an encoding for ladder-lotteries.

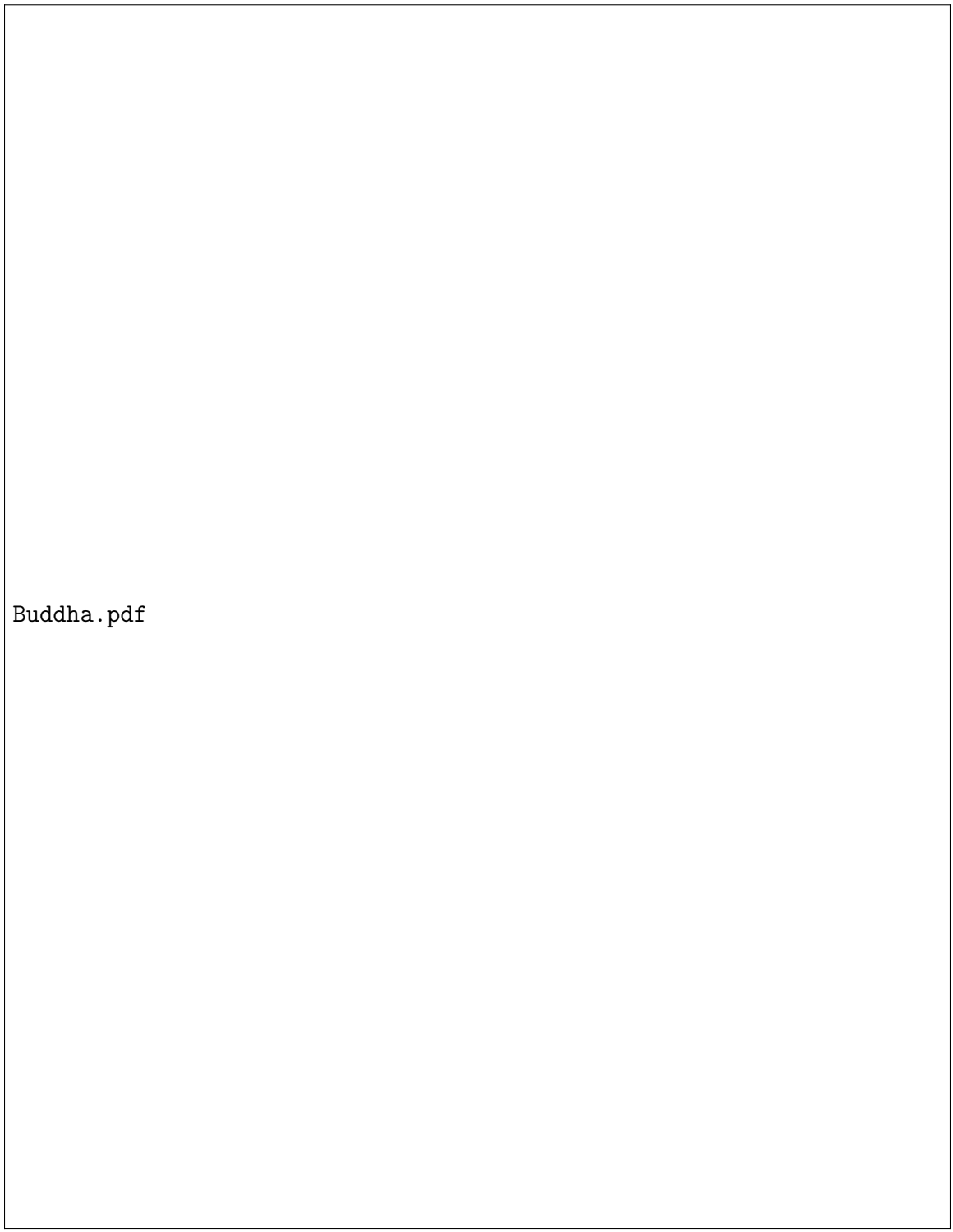


Figure 1.1: A picture of Buddha Amithaba

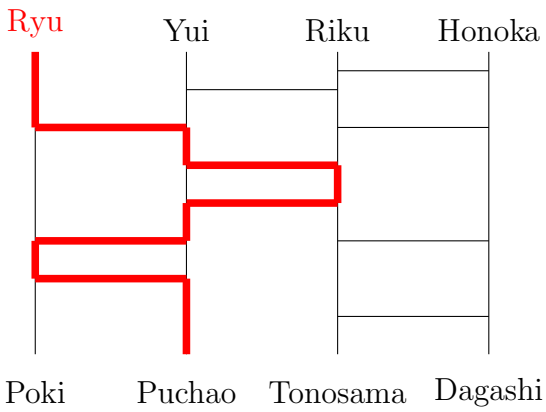


Figure 1.2: A ladder lottery where Ryu gets Puchao, Yui gets Dagashi, Riku gets Tonosama and Honoka gets Poki. You can see that Ryu's path is marked by red bars.

1.2 Overview of Thesis

This thesis is broken down into several sections. Firstly, an introuduction to Amidakuji, and how they pertain to computer science will be presented. This will be followed by a literature review of ladder lotteries in which discussions of solved problems will be provided, along with the commonalities between ladder lotteries and other mathematical objects. Following the literature review, the methodology used to conduct the research will be discussed. This section will focus primarily on the algorithms used to generate the data for this reasearch. Following the methodology section, the findings of the research will be discussed and analyzed. In this section there will be proofs and formulas for certain propositions made about ladder lotteries. This section will contain the bulk of the findings for this thesis. Following the results section, a summary of future work will be provided. In this section, the failures and successes of this research will be analyzed. There will also be commentary on open (unsolved) problems related to ladder lotteries and a discussion of how research on ladder lotteries could be used in other fields. Finally, a conlcusion that summarizes the thesis will be provided.

Chapter 2

Background

An interesting property about ladder lotteries is that they can be derived from a *permutation* which is a is a unique ordering of objects. [**?**] For the purposes on this paper, the objects of a permutation will be integers ranging from $[1 \ldots N]$. *Optimal ladder lotteries* are a special case of ladder lotteries in which there is one bar in the ladder for each *inversion* in the permutation [**?**]. An *inversion* is a relation between two elements in π, π_i and π_j, such that if $\pi_i > \pi_j$ and $i < j$ then π_i and π_j form an inversion. For example, given $\pi = (4, 3, 5, 1, 2)$, its iversion set is $Inv(\pi) = \{(4, 3), (4, 1), (4, 2), (3, 1), (3, 2), (5, 1), (5, 2)\}$. Every permutation has a unique, finite set of optimal ladder lotteries associated with it. Thus, the set of optimal ladder lotteries associated with π, hereby known as $OptL\{\pi\}$, is the set containing all ladder lotteries with a number of bars equal to the number if inversions in π. See Fig. 2.1 for an example of an optimal ladder in $OptL\{(4, 3, 2, 1)\}$. For each optimal ladder in $OptL\{\pi\}$, the N elements in π are listed at the top of a ladder and each element is given its own line. At the bottom of a ladder is the *sorted permutation*, hereby known as the *identity permutation* [**?**]. The identity permutation of size N is defined as follows - $I : (1, 2, 3, \ldots, N)$. Each ladder in $OptL\{\pi\}$ has the minimal number of horizontal bars to sort π into the identity permutation. Each bar in a ladder from $OptL\{\pi\}$ uninverts a single inversion in π exactly once. For the remainder of this paper, only optimal ladder lotteries will be discussed, with one exception. Therefore when the term ladder lottery is used, assume optimal ladder lottery unless otherwise stated.

Figure 2.1: Two ladders for the permutation (4, 3, 2, 1). The left ladder is an optimal ladder and the right ladder is not. Therefore the left ladder belongs to $optL\{(4,3,2,1)\}$. The bold bars in the right ladder are redundant, thus the right ladder is not optimal

2.1 Literature Review

The study of ladder lottieres as mathematical objects began in 2010, in the paper **Efficient Enumeration of Ladder Lotteries and its Application**. The paper was written by four authors, Yamanaka, Horiyama, Uno and Wasa. In this paper the authors present an algorithm for generating all the ladder lotteries of an arbitrary permutation, π. Since this paper emerged, there have been several other paper written directly about ladder lotteries. These papers include **The Ladder Lottery Realization Problem**, **Optimal Reconfiguration of Optimal Ladder Lotteries**, **Efficient Enumeration of all Ladder Lotteries with K Bars**, **Coding Ladder Lotteries** and **Enumeration, Counting, and Random Generation of Ladder Lotteries**.

2.2 Efficient Enumeration of Laddder Lotteries and its Application

In their paper, Efficient Enumeration of Ladder Lotteries and its Application, the authors provide an algorithm for generating $OptL\{\pi\}$ for any π, in $\mathcal{O}(1)$ per ladder [**?**].

This is the first and only published algroithm for generating $OptL\{\pi\}$. The paper also presents the number of ladder lotteries in $OptL\{(11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)\}$ which is $5,449,192,389,984$ [**?**]. This is a very impressive accomplishment for reasons which will be discussed later in the literature review.

The authors' algorithm is based on several key concepts, the most important of which is the *local swap operation*. This is the minimal change operation that transitions from one ladder in $OptL\{\pi\}$ to the next ladder. The local swap operation is essentially a 180 degree rotation of three bars in the ladder, such that the bottom bar is rotated to the top, the middle bar stays in the middle and the top bar is rotated to the bottom. If the bars undergo a 180 degree rotation to the right, then this is known as a *right swap operation* and if the bars udergo a 180 degree rotation to the left then this is known as a *left swap operation*. To go to the next ladder in the set, the current ladder, L_i udergoes a right swap operation to get to ladder L_{i+1}. See Fig. 2.2 for an exmaple of a local swap operation.The *route* of an element is the sequence of bars in the ladder that an element must cross in order to reach its correct position in the identity permutation. The sequence is ordered from top left to bottom right. Note that each bar has two elements that cross it, therefore the bar belongs to the route of the greater of the two elements. It is important to note that when a right swap operation occurs, two of the three bars belong to the route of a greter element and one bar belongs to the route of a lesser element. Once rotated, the bar of the lesser element is above the bars of the greater element.

The *clean level* refers to the smallest element in π such that none of its bars have undergone a right swap operation. If there is no such element, then the clean level is the maximum element in $\pi + 1$. The *root ladder* is the only ladder in the set with a clean level of 1; in other words, the root ladder is the only ladder in which no bars have undergone a right swap operation. The root ladder is unique to $OptL\{\pi\}$. To see the root ladder of $OptL\{(4, 5, 6, 3, 1, 2)\}$ please refer to figure Fig. 2.3. Since none of the bars in the root ladder have undergone a right swap operation, it is the only ladder

in $OptL\{\pi\}$ that has a clean level of 1. The root ladder is also the original decendant ladder in the set. Insofar as the enumeration algorithm is based on performing a right swap operation on a pervious ladder, then every other ladder must have at least one right swap operation. Since the root ladder has no right swap operations, then it must be the decendant of every other ladder.

The algorithm for generating $OptL\{\pi\}$ in the paper was the backbone of the research for this thesis. However, the algorithm in this paper had some issues. These issues presented several challenges during the research for this thesis. The first issue in the paper is that the authors do not provide an algorithm for generating the root ladder in $OptL\{\pi\}$. Seeing as every other in the set is derived from the root ladder, it is essential to build the root ladder without performing a right swap operation. Yet the paper does not provide such an algorithm. The second issue in this paper is that there is no algorithm for permforming a local swap operation on a ladder. Although the authors do a good job explaining under what conditions a local swap operation can be performed [**?**], the actual operation itself is trickier than it seems. The last issue in the paper is that it contains an error. The error is that one of the diagrams is incorrect. The diagram is of all the ladders in $OptL\{(5, 6, 3, 4, 2, 1)\}$. This diagram contains 76 ladders when there are actually only 75. The error was confirmed by the author Yamanaka in an email correspondence he and I had. These issues will be resolved in the Methodology and Implementation section.

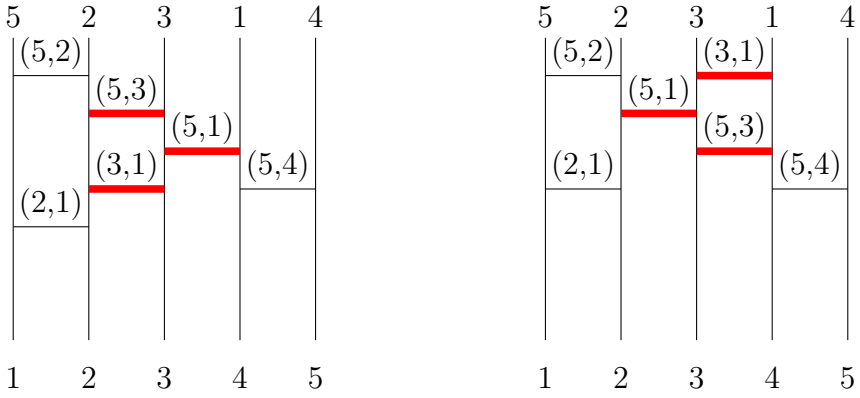


Figure 2.2: Example of a local swap operation. When a right swap operation is permformed on the left ladder, the result is the right ladder. When a left swap operation is permformed on the right ladder, the result is the left ladder.

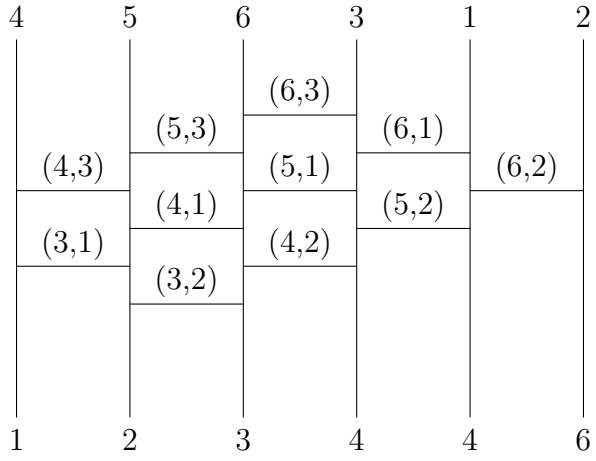


Figure 2.3: The root ladder for $OptL\{(4,5,6,3,1,2)\}$. Notice how none of the bars have undergone a right swap operation. This is clear when considering that there is no bar of a lesser element above the bar(s) of a greater element.

2.3 Ladder-Lottery Realization

In their paper **Ladder-Lottery Realization** the authors provide a rather interesting puzzle in regards to ladder lotteries. The puzzle is known as the ladder-lottery realization problem [**?**]. In order to understand the problem, one must know what a *multi-set* is. A *multi-set* is a set in which an element appears more than once. The exponent above the element indicates the number of times it appears in the set. For example, given the following multi-set, $\{3^2, 2^4, 5^1\}$ the element 3 appears twice in the set, the element 2 appears four times in the set and the element 5 appears once in the set. The ladder-lottery realization puzzle asks, given an arbitrary starting permutation and a multi-set of bars, is there a *non-optimal* ladder lottery for the arbitrary permutation that uses every bar in the multi-set the number of times it appears in the multi-set [**?**]. For an example of an affirmative solution to the ladder lottery realization problem, see Fig. 2.4.

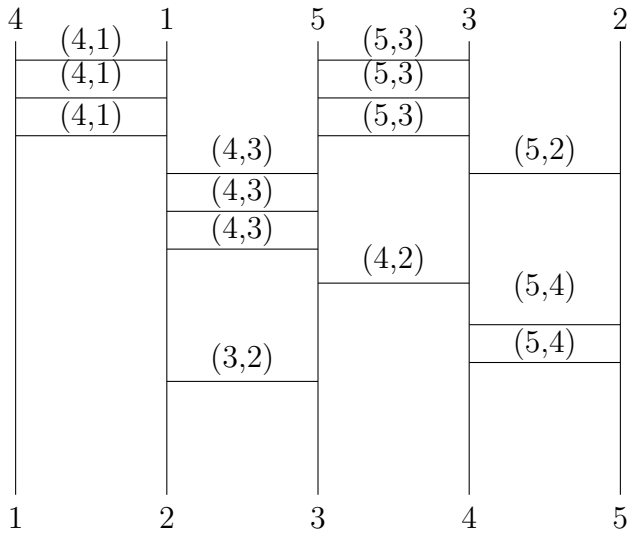


Figure 2.4: An affirmative solution to the Ladder Lottery Realization Problem given a starting perumtation $(4, 1, 5, 3, 2)$ and the multi set of bars $\{(4, 1)^3, (4, 3)^3, (4, 2)^1, (5, 4)^2, (5, 3)^3, (5, 2)^1, (3, 2)^1\}$

The authors prove that the ladder-lottery realization problem in NP-Hard by reducing the ladder-lottery realization to the One-In-Three 3SAT, which has already been proven to be NP-Hard [**?**]. The One-In-Three 3SAT problem is a problem such that given a set of variables (X), a collection of *disjuntcive clauses* (C) which are disjunctive expressions over literals of X. Each clause in C must contain three literals then is there a truth assignment for X such that each clause in C has exactly one true literal. For eaxmple, let $X = \{p, q, r, s, t\}$ and let $C = \{C_{p,q,s}, C_{r,q,s}C_{p,s,t}, C_{r,t,q}\}$, the question is whether it is possible for each clause to have exactly one true literal. The answer in this case is yes. If $p = T$, $r = T$, $q = F$, $s = F$ and $t = T$ then all the clauses in C have exactly one true literal. The authors reduce the ladder lottery-realization problem to the One-In-Three 3SAT problem by devising four gadgets [**?**]. The result of the reduction is that the arbitrary starting permutation is equivelent to a derivation of the intial set of variables, X, in the One-In-Three 3SAT problem and the multi-set of bars is equivelent to a derivation of the intial set of clauses, C, in the One-In-Three 3SAT problem citeA3.

The authors note that there are two cases in which the ladder-lottery realization problem can be solved in polynomial time. These cases include the follwing. First, if every bar in the multi-set appears exactly once and every bar corresponds to an inversion, then an affirmative solution to the ladder-lottery realization instance can be demonstrated in polynomial time [**?**]. Second, if there is an inversion in the perumutation and its bar appears in the multi-set an even number of times, then a negative solution to the ladder-lottery realization instance can be solved in polynomial time [**?**].

2.4 Optimal Reconfiguration of Optimal Ladder Lotteries

In **Optimal Reconfiguration of Optimal Ladder Lotteries**, the authors provide a polyminomial solution to the minimal reconfiguration problem. The prob-

lem states that given two ladder is $OptL\{\pi\}$, L_i and L_m, what is the minimal number of local swap operations to perform that will transition from L_i to L_m [**?**]. The authors do so based on the local swap operations previously discuessed along with some other concepts. The first of these concepts is termed the *reverse triple*. Basically, a reverse triple is a relation between three bars, x, y, z in two arbitrary ladders, L_i, L_m, such that if x, y, x are right rotated in one of the ladders, then they are left rotated in the other. The second of the concepts is the *improving triple*. The improving triple is essentially a bar that can be left/right rotated such that the result of the rotation the bar removes a reverse triple between two arbitrary ladders L_i and L_m [**?**]. The solution to transition from L_i to L_m with the minimal length reconfiguration sequence is achieved by applying improving triple to the reverse triples between L_i and L_m. That is to say, the length of the reconfiguration sequence is equal to the number of reverse triples between L_i and L_m [**?**].

The second contribution of this paper is that it provides a closed form upper bound for the minimal length reconfiguration sequence for any permutation of size N. That is to say, given any permutation, π, of size N what is the maximum length of a minimal reconfiguration sequence between two ladders in $OptL\{\pi\}$. The authors prove that it is $OptL\{\pi_{N,N-1,\ldots,1}\}$ that contains the upper bound for the minimal length reconfiguration sequence between two ladders L_i and L_m [**?**]. Moreover, it is only the root ladder and terminating ladder in $OptL\{\pi_{N,N-1,\ldots,1}\}$ whose minimal reconfiguration sequence is equal to the upper bound. That upper bound is $N\binom{(N-1)}{2}$. This is because the number of reverse triples between the root ladder and the terminating ladder in $OptL\{\pi_{N,N-1,\ldots,1}\}$ is equal to $N\binom{(N-1)}{2}$. Thus, in order to reconfigure the root to the terminating ladder, or vice versa, each reverse triple between them must be improved.

2.5 Efficient Enumeration of all Ladder Lotteries with K Bars

In this paper, the authors apply the same algorithm used in Efficient Enumeration of Optimal Ladder-Lotteries and its Application for generating all ladder lotteries with k bars [**?**]. The number of elements in The inversion set of π also known as $Inv\{\pi\}$ provides the lower bound for K and the upper bound is positive infinity. Therefore $K = [|Inv\{\pi\}| \dots N]$ [**?**].

2.6 Coding Latter Lotteries

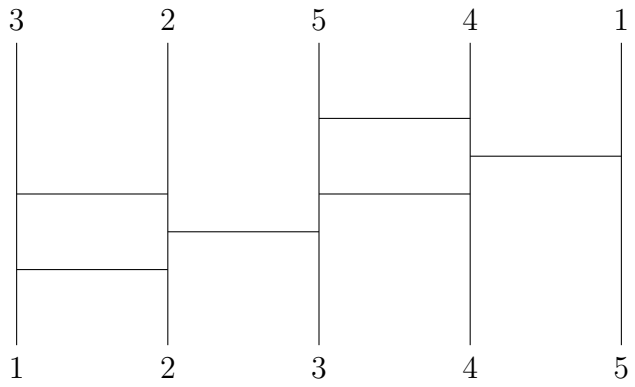
2.6.1 Overview

In this paper, the authors provide three methods to encode ladder-lotteries as binary strings. Coding discrete objects as binary strings is an appealing theme because it allows for compact represntation of them for a computer [**?**].

2.6.2 Route Based Encoding

The first method is termed *route based encoding method* in which each route of an element in the permutation has a binary encoding. Let L_k be a ladder lottery for some arbitrary permutation $\pi = (p_1, \dots, p_n)$. The route of element p_i is encoded by keeping in mind p_i crosses bars in its route going left zero or more times and crosses bars in its route going right zero or more times [**?**]. The maximum number of bars p_i can have is $n-1$, therefore the upper bound for the number of left/right crossings for p_i is $n-1$ [**?**]. Let a left crossing be denoted with a $'0'$ and let a right crossing be denoted with a $'1'$. Let C_{pi} be the route encoding for the i^{th} element in π. To construct C_{pi}, append 0 and 1 to each other representing the left and right crossings of p_i from the top left to bottom right of the ladder [**?**]. If the number of crossings for p_i is less than $n-1$, append $0s$ to the encoding of the route of p_i until the encoding

is of length $n-1$ [**?**]. Let LC_L be the route encoding for some arbitrary ladder in $OptL\{\pi\}$ is $C_{p_1}, C_{p_2,\ldots C_{p_N}}$. For an example of the route encoding for the root ladder of $(3,2,5,4,1)$ refer to Fig. 2.5. In Fig 2.5 you will see that C_{p_1} is $11\underline{00}$. Underlined $0s$ are the $0s$ added to ensure the length of C_{p_1} is $N-1$. Since the length of C_{pi} is $n-1$ and the number of elements in π is n then the length of $LC_L = n(n-1)$. Hence the number of bits needed for LC_L belongs to $\mathcal{O}(n^2)$.

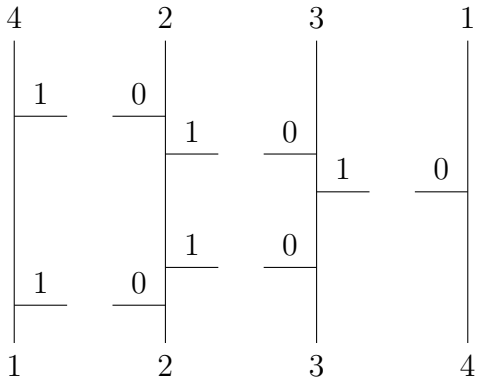


2.6.3 Line Based Encoding

The second method is termed *line based encoding* which focuses on encoding the lines of the ladder-lottery. Each line is represented as a sequence of endpoints of bars. Let L be an optimal ladder-lottery with n lines and b bars, then for some arbitrary line i there are zero or more right/left endpoints of bars that come into contact with line i [**?**]. Let lc_i denote the line based encoding for line i. Let 1 denote a left end point that comes into contact with line i and let 0 denote a right end point that comes into contact with line i. Finally, append a 0 to line i to denote the end of the line. Then line i can be encoded, from top to bottom, as a sequence of $1s$ and $0s$ that terminates in a 0. Given the ladder in Fig. 2.5, lc_3 is $001\underline{0}$. The $\underline{0}$ denotes the end of the line. Let LC_L be the line encoding for some arbitrary ladder, then $LC_L = lc_1, lc_2, \ldots lc_n$. Let $L_{2.5}$ refer to the ladder in Fig. 2.5, then $LC_{L_{2.5}} = 11\underline{0}010\underline{0}110\underline{0}010\underline{0}0\underline{0}$

In order to reconstruct L_k from LC_{L_k}, or in other words decode LC_{L_k} it is

important to recognize that the first line only has left endpoints attached to it [**?**]. Since left end points are encoded as a 1 then it is guarenteed that the first 0 represents the end of line 1. Secondly, the last/nth bar has only right end points attached to it. Therefore lc_n will only have $0s$. Therefore, lc_n does not require a terminating 0. Thirdly, for any line $i+1$, if line $i+1$ has a 0 then there must be a corresponding 1 in line i. That is to say, if the right end point of a bar is on line $i+1$ then that same bar must have a left endpoint on line i. To decode LC_L start by decoding line 1. The line will contain 0 or more left end points. To decode lc_{i+1} where $i+1>1$, go to lc_i and match each 1 in lc_i with a 0 in lc_{i+1}. Let $k=$ the number of $1s$ in lc_i. Let $j=$ the number of $0s$ in lc_{i+1} then $k=j-1$; due to the last 0 in lc_{i+1} denoting the end of line $i+1$. Intuitively, this means match every left end point of a bar in line i with a right end point in line $i+1$. The last 0 represents the end of line $i+1$. For the $1s$ in lc_{i+1} draw a left end point on line $i+1$ relative to where the 1 occured to its left and right neigbor in lc_{i+1}. For an example of a full decoding of $LC_{L_{(4,2,3,1)}}$ please refer to Fig. 2.6.



Since each bar is encoded as two bits, and there are $N-1$ bits as terminating bits; one for each line in L, then the number of bits required is $N+2B-1$, where N is the number of lines and B is the number of bars. Encoding and decoding can be done in $\mathcal{O}(n+b)$ time. Clearly the line-based encoding trumps the route-based encoding in both time and space complexity.

2.6.4 Improved Line-Based Encoding

Although the line-based encoding is better than the route based encoding, it can still be further optimized. The authors provide three improvements to the line-based encoding. These three improvements can be combined to really help imrpove the line based encoding's space efficiency [**?**].

Imrpovement 1

Since the nth line has only right endpoints attached to it, then it actually does not need to be encoded. Right endpoints are denoted as 0 and left endpoints are encoded as 1, therefore the number of right endpoints for line n is equal to the number of $1s$ in lc_{n-1}. Thus, there is no need for lc_n [**?**]. The encoding with improvment one for the ladder in Fig. 2.6 is 11$\underline{0}$0110$\underline{0}$010.

Improvement 2

Improvement Two is based off of the fact that given any two bars, x, y let l_x denote the left endpoint of bar x, let l_y denote the left endoint of bar y, let r_x denote the right end point of bar x and let r_y denote the right end point of bar y. Let line i be the line of l_x and l_y and let line $i+1$ be the line of r_x and r_y.

Theorem 2.6.1 *There are three possible cases for the placement of x and y in some arbitrary ladder from Opt$L\{\pi\}$. The first case is that there is at least one other bar, z, with a right end point, r_z between l_x and l_y on line i. The second case is that there is at least one other bar z, with a left end point, l_z, between r_x and r_y on line $i+1$. The third case is that there is at least one bar, z, with a right end point, r_z, betwen l_x and l_y on line i and there is at least one other bar, $z\prime$ with a left end point, $l_{z\prime}$, between r_x and r_y on line $i+1$ [?]. For an example of all three cases refer to Fig 2.7.*

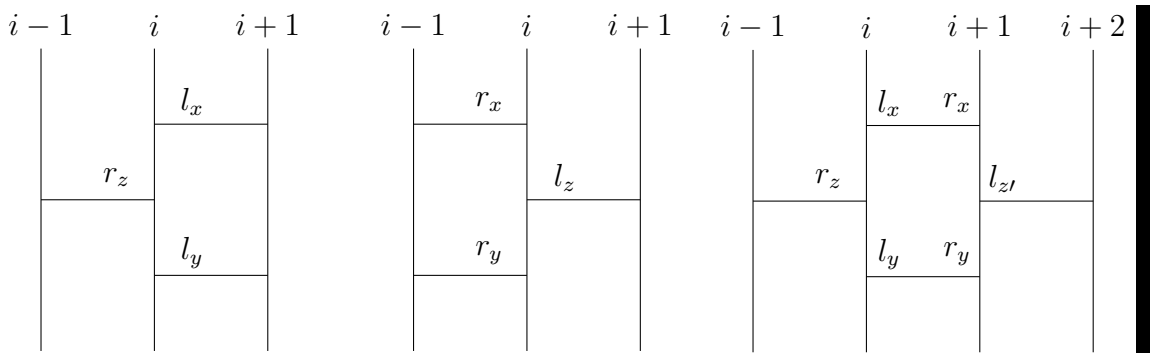


Figure 2.5: Three examples of the three cases for the placement of bars x and y in a ladder-lottery

Proof. Suppose that none of the above cases hold. Let L_π be an optimal ladder-lottery with bars x and bar y. If none of the cases hold then x and y are directly above/below each other without the enpoint of some third bar z between l_x and l_y or between r_x and r_y. Let x be the bar for the inversion of two elements p and q in π. As p and q travel through the ladder they will cross each other at bar x; thus uninverting them. Since bar y is directly below bar x, then p and q will cross bar y

thus re-inverting them. Therefore, there will need to be a third bar that uninverts p and q a second time. Since this third bar is redundant, L_π is non-optimal which is a contradiction. Let x be a bar for two elements in π, p and q such that p and q do not form an inversion. Then x will invert p and q and y will uninvert them. Thus making both x and y redundant bars which is also a contradiction. Therefore one of the above cases must hold. □

Knowing that one of the three above cases must hold is beneficial for improving the line-based encoding. If l_x and l_y on line i have no r_z between them, then there must be at least one $l_{z'}$ between r_x and r_y on line $i+1$. Since a left endpoint is encoded as a 1 and a right endpoint is encoded as a 0, a 1 can be omitted for the encoding of line $i+1$ if l_x and l_y have no r_z between them on line i [**?**]. That is to say, if there is not a 0 between the two $1s$ for l_x, l_y in lc_i, it is implied that there is at least one 1 between the two $0s$ for r_x, r_y on lc_{i+1}. Hence, one of the $1s$ in lc_{i+1} can be omitted. The line encoding with improvement two for the ladder in Fig 2.6 is 11$\underline{0}$010$\underline{0}$00$\underline{0}$0.

Imrpovement 3

Improvement three is based off of saving some bits for right end points/$0s$ in lc_{n-1}. Since line n has no left end points, then then there must be some right endpoints between any two consecutive bars connecting lines $n-1$ and line n. If you refer to Fig. 2.7, then the only configuration for lines $n-2, n-1, n$ is the middle configuration [**?**]. Knowing this, then given two bars, x and y with l_x/l_y on line $n-1$ and r_x/r_y on line n, there must be at least one bar, z, with its r_z between l_x and l_y on line $n-1$. Thus, for every 1 in lc_{n-1} except the last 1 in lc_{n-1}, a 0 must immidediately proceed any 1 in lc_{n-1}. Since this 0 is implied, it can be removed from lc_{n-1} [**?**]. For an example of improvement three with its line encoding for lc_{n-1} please refer to Fig. 2.8.

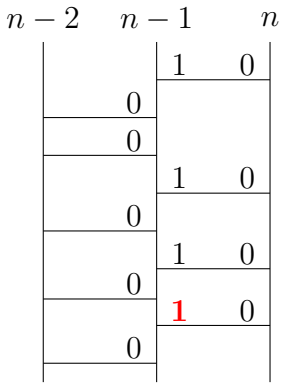


Figure 2.6: The line coding for lc_{n-1} with imrpovement three is 1011100. As always, 0 denotes the end of the line encoding. The red, bold 1 represents the last left end point in lc_{n-1}, therefore the proceeding 0 must be included in lc_{n-1}. For every other 1 in lc_{n-1}, a 0 is omitted following said 1.

Combining All Three

The combination of all three improvements can be done independently. Let IC_L be the *improved line-based encoding* for some ladder L by applying improvements 1-3 to LC_L. Recall that LC_L denotes the line-based encoding for some ladder L. LC_L for the ladder in Fig. 2.9 is 110101010010101000. By applying imrpovement one, we get 110101011000101010. Notice how the last three 0s from LC_L were removed because they represented lc_n. By applying imrpovement two to improvememt one we get 110100110010010. Notice how the second, and eigth 1 were removed because they are implied by the successive 0s. By applying improvement three to the result of improvement two we get 110100110001010. Notice how the last 0 was removed from improvement two. This is because the 0 implied in lc_{n-1} due to the configuration between of bars connecting line $n-1$ and line n. Thus, IC_L for Fig. 2.9 is $IC_{2.9}$ = 110100110001010.

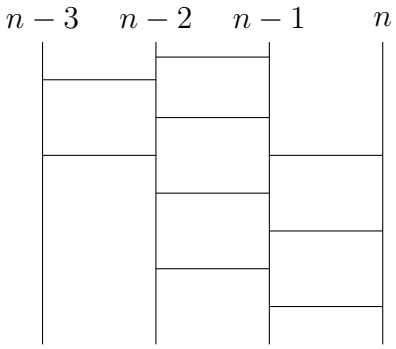


Figure 2.7: A ladder used to illustrate all three improvements IC_L. IC_L = 11$\underline{0}$10011$\underline{0}$001010$\underline{}$

2.7 Enumeration, Counting, and Random Generation of Ladder Lotteries

In this paper, the authors consider the problem of enumeration, counting and random generation of ladder-lotteries with n lines and b bars [**?**]. It is important to note that the authors considered both optimal and non-optimal ladders for this paper. Nonetheless, the paper is still fruitful for its modelling of the probelems and insights into ladder-lotteries. The authors use the line-based encoding, LC_L for the representation of ladders that was discussed in the review of **Coding Ladder Lotteries**.

2.7.1 Enumeration

The authors denote a set of ladder lotteries with n lines and b bars as $S_{n,b}$. The problem is how to enumerate all the ladders in $S_{n,b}$ [**?**]. The authors use a *forest structure* to model the problem. A *forest structure* is a set of trees such that each tree in the forest is dijoint union with every other tree in the forest. Consider $S_{n,b}$ to be tree in the forest. That is to say, a union disjoint subset of all ladders with n lines

and n bars. Then $F_{n,b}$, or the forrest of all $S_{n,b}s$ is the set of all ladders with n lines and n bars [**?**]. For an example of a forest for $F_{3,2}$ refer to Fig. 2.10

The authors create $F_{n,r}$ by defining a removal sequennce for each LC_L [**?**]. Each ladder, L, in $F_{n,r}$ is a leaf node. By removing the second last bit of LC_L the result is $P(LC_L)$ and the resulting substructure is some *sub-ladder*, $P(L)$, which is an incomplete ladder containing unmatched endpoints of bars or a missining line [**?**]. For example, given $LC(L) = 10100$, $P(LC_L) = 1010$. Notice how the second last bit was removed. By removing the second last bit from $P(LC_L)$ we get $P(P(LC_L))$ and $P(P(L))$ respectively. The removal sequence is repeated until the sub-ladder consists of two lines with 0 endpoints attached to line 2 and 0 to r left endpoints are attached to line 1. There are $r + 1$ terminating sub-ladders, i.e., roots of trees in $F_{n,r}$.The removal sequence is uniqe for each ladder in $F_{n,r}$ is unique.

2.7.2 Counting

The authors provide a method and algorithm to count all ladders with n lines and b bars. According to the authors, the enumeration algorithm is much slower than the counting algorithm [**?**]. The counting algorithm works by dividing ladders into four types of sub-ladders. For sub-ladder, R, its type is a tuple $t(n, h, p, q)$ where n is the number of lines, h is the number of half bars, p is the number of unmatched end-points on line $n-1$ and q is the number of unmatched end-points on line n. From this type there are four sub-divisions of sub-ladders.

Case 1: $h < p + q$ **or** $n < 2$

There are zero ladders because it is impossible for the root sub-ladder to have less than two lines. It is also impossible for the number of half bars, h, to be less than the number of detached left end points on line $n-1$ plus the number of detached end points on line n.

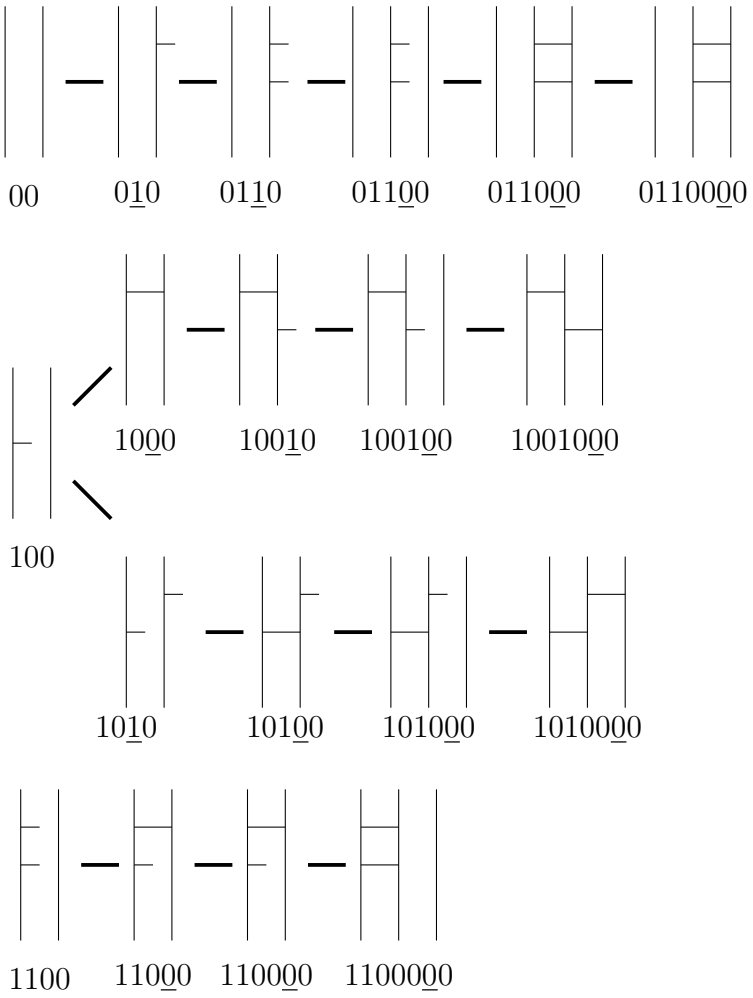


Figure 2.8: The forrest, $F_{3,2}$ where 3 is the number of lines and 2 is the number of bars. All ladders with 3 lines and 2 bars are leaf nodes of one of three trees $S_{3,2}$. The underlined bits are the inserted second last bit from the parent's line-encoding resulting in the child's line encoding

Case 2: $n = 2$ **and** $h = p$ **and** $q = 0$

There is only one ladder because the number of half bars on the last/$2nd$ line is 0 since $q = 0$. Therefore all half bars are on the $n - 1th/1st$ line of the sub-ladder. This is known because $h = p$ which means the number of half bars is the same as the number of unmatched bars on line $n - 1/1st$ Hence, the unmacthed half bars on the $1st$ line must be connected to the $2nd$ line. Once these are all matched the ladder will be complete. Thus, there is only one ladder for this case.

Case 3: $(n \geq 3$ **or** $h > p)$ **and** $q = 0$

If this is the case, then there are no endpoints attached to line n, but the number of half bars is greater than the number of enpoints attached to line $n-1$, which means there is some line(s) $n - t$, $t > 2$ that have end points attached to them. Let R be a sub-ladder of type $R = t(n, h, p, q)$ with the the above values for n, h, p, q. Let $P(R)$ be R with the removal of $R's$ second last bit in LC_R; i.e. the parent of R. The LC_R must have a 0 for the second last bit. This 0 designates either the end of line $n - 1$ or a right endpoint of a bar attached to line $n - 1$. If the second last bit in LC_R is the right end point of some bar, then $P(R) = t(n, h-1, p+1, q)$. This is because the $n-1th$ bar has a right end point that must be connected to some left endpoint at line $n-2$. Since the removal sequence of the second last bit ensures that there cannot be a right end-point detached from a left end-point. Only left end-points can be detached from right end-points [**?**]. However, if the second last bit of LC_R designates the end of line $n - 1$, then $P(R) = t(n - 1, h, 0, p)$. This is because the removal of the second last bit is the removal of the end of line $n - 1$ in R. Thus, line n must be empty in R since the last bit in LC_R designated the end of line n. Thus, if line n is empty and the end point of line $n-1$ has been removed from LC_R, resulting in $P(LC_R)$, the last bit in $P(LC_R)$ must be the end of line $n - 1$ in R resulting in a pre-ladder with one less line than R.

In order to count the number of ladders of type $t(n \geq 3, h > p, q = 0)$ the

authors demonstrate an injection from $t(n \geq 3, h > p, q = 0)$ to $t(n-1, h, 0, p) \cup t(n, h-1, p+1, q)$ [**?**]. They then demonstrate that the $|t(n \geq 3, h > p, q = 0)| = |t(n-1, h, 0, p)| + |t(n, h-1, p+1, q)|$.

Case 4: $h \geq p + q$ **and** $q > 0$

Let R be a pre-ladder of type $t(n, h, p, q)$. Then the second last bit of LC_R is either a 0 or a 1. If it is a 0 then it represents a right end point attached to line n. Thus, removing it to get $P(LC_R)$ is in effect detaching a right end point from some left end point on line $n-1$. Therefore, the parent, $P(R)$ is of type $t(n, h-1, p+1, q)$. Seeing as in the parent, there is now a left end point detached from its right end point in R. However, if the second last bit of LC_R is a 1, then this indicates the left half of a bar on line n. But since there is no bar $n+1$, this left end point must be detached. Therefore, by removing this 1 in LC_R results in a parent with one less detached end point on line n. Thus $P(R)$ is of type $t(n, h-1, p, q-1)$. This leads the authors to conclude $|t(n, h \geq p+q, q > 0)| = |t(n, h-1, p+1, q)| + |t(n, h-1, p, q-1)|$ [**?**].

2.7.3 Random Genearation

The random generation of ladder lotteries with n lines and b bars is done by the recurrence relations in the counting and enumerating sections. The goal is to produce some L of type $t(n, 2b, 0, 0)$ where the number of half bars equals the total $2(b)$ and there are no detached end points on lines $n-1$ and n. This implies that there are no detached endpoints on any line $n-t$ where $t \geq 2$ because the removal sequence from the $LC_{pre-ladder}$ ensures that any line before $n-1$ has no detached endpoints. Thus, if L is of type $t(n, 2b, 0, 0)$ it is no longer a pre-ladder but a complete ladder with n lines and b bars [**?**].

The authors use an algorithm to generate a random integer, x, in $[1, |t(n, h, p, q)|]$. where $t(n, h, p, q)$ corresponds to some parent type of ladder. $t(n1, h1, p1, q1)$ corresponds to one child type of $t(n, h, p, q)$ and $t(n2, h2, p2, q2)$ corresponds to the other

child type. If $x \leq |t(n1, h1, p1, q1)|$ then generate a pre-ladder of type $t(n1, h1, p1, q1)$ else generate a pre-ladder of type $t(n2, h2, p2, q2)$ [**?**]. Continue until there is type $t(n, 2b, 0, 0)$ which corresponds to a complete ladder lottery with n lines and b bars.

Chapter 3

Methodology and Implementation

3.1 The Gray Code Problem

3.1.1 Introduction to the Problem

Gray Codes are enumerations of objects in a set such that there is minimal amount change to transition from Obj_i to Obj_{i+1}. In order to define the mininmal amount change, a basic operation needs to be defined that constitutes a change from one object to the next. For example, enumerating all binary strings of length N defines the basic operation as flipping a bit. The better the Gray Code, the lower the number of bit flips are required to transition from $BinaryString_i$ to $BinaryString_{i+1}$. The problem of finding a Gray Code for generating canonical ladders from $OptL\{\pi\}$ is inspired by finding the most efficient way to produce a single ladder acting as a representative for all $OptL\{\pi_n\}$. The formal statement of the problem is the following; given a value $N \geq 3$ where $N \in Z$, generate a canonical ladder from $OptL\{\pi_N\}$ for each π of order N. For example, let $N = 4$, then there are twenty-four, or $N!$ permutations of order N. These permutations, listed lexicographically (smallest to largest) are:

1234 1243

1324 1342

1423 1432

2134 2143

2314 2341

2413 2431

3124 3142

3214 3241

3412 3421

4123 4132

4213 4231

4312 4321

Each of these permutations has zero or more ladders in each of their respective $OptL\{\pi\}$. The Gray Code problem asks, what is the most efficient way to enumerate a canonical ladder from each $OptL\{\pi_i\}$ such that a minimal change from one canonical ladder can be applied to said ladder resulting in the canonical ladder in $OptL\{\pi_{i+1}\}$. In this thesis, four Gray Codes were used to generate the canonical ladders for each $OptL\{\pi_N\}$. Each of these Gray Codes are modifications of already existing Gray Codes for generating all permutations of order N. The Gray Codes used are Steinhaus-Jonson-Trotter Gray Code, the Zaks Gray Code, Heaps Gray Code and lexicographic Gray Code. Unlike with most Gray Codes, there are two basic operations defined for the modified Gray Codes for enumerating canonical ladders from $OptL\{\pi\}$. The first basic operation is the insertion or deletion of a bar. The second operation is the swap of two bars.

Theorem 3.1.1 *The first basic operation is necessary to transition from L_i to L_{i+1} whereas the second operation is not.*

Proof. Suppose that the first of the two basic operations was not necessary. Let L_i be the canonical representative of $OptL\{\pi_i\}$ of order N. Let L_{i+1} be the canonical representative of $OptL\{\pi_{i+1}\}$ of order N. Let x be any bar in L_i. Note that x represents an inversion in π_i. If x is not removed from L_{i+1} then L_{i+1} must have an additional bar, otherwise π_i and π_{i+1} are the same permutation. Thus, at least one new bar, y is inserted to L_{i+1}. Suppose that x is removed L_{i+1}, then a removal operation has been performed in order to transition from L_i to L_{i+1}. In both cases

the first basic operation is performed. If neither of the above cases are true, then $L_i = L_{i+1}$ which is a contradiction. □

Proof. Suppose that the second basic operation is necessary. Consider the ladder for the idendity permutation of order N. Let L_{Id} denote this ladder. Seeing as it is the only ladder in $OptL\{\pi_{Id}\}$ then it must be the canonical representative. Let L_{i+1} be the canonical representative of the next permutation's $OptL\{\pi\}$. Seeing as L_{Id} has no bars to swap, then it is impossible to derive L_{i+1} from L_{Id} by performing a swap operation. Thus, it is not necessary to permform a swap operation to transition from a canonical representative to the next representative. In fact, in the above example it is impossible to perform a swap operation. □

The canonical representative was chosen in two ways. The first way of chosing the canonical representative was to always choose the root ladder from each $OptL\{\pi_N\}$. The rationale for always choosing the root ladder was due to the fact that every permutation has a root ladder in its $OptL\{\pi\}$ whereas not every permutation has a non-root ladder. For example, given the permutation $(2, 1, 4, 3)$, $OptL\{(2, 1, 4, 3)\}$ has only one ladder which is the root ladder. This is because there are only two bars in the ladder, thus there is no swap operation that can be performed.

Theorem 3.1.2 *If $|OptL\{\pi\}| = 1$ then the ladder is the root ladder.*

Proof. The root ladder is defined as the ladder whose clean level is one. This means either there is no bar of a lesser element above the route a greater element. Keeping in mind that the clean level of the root ladder is one, next consider what is meant by a *child bar* which is a bar to the bottom left or right of a given bar x. Within the context of the root ladder, if the left endpoint of the child bar is directly below the right end point of x then the child is a right child of x. If the right end point of the child bar is directly below the left end point of x then it is a left child. Keeping in mind the root ladder has not undergone any right swap operations, then if a child is a right child then the child belongs to the same route of x in the root ladder. Let R_m

denote this route. Let x be a bar representing an inversion with element m and k. The right child of x is a bar which represents an inversion with m and some element to the right of k. Suppose this was not the case, then this would mean that the right child of x was either a bar representing an inversion between some element m' that was greater than m or lesser than m. If m' was greater than m then this would be a contradiction seeing as x would be above the bar of a route of a greater element which contradicts the definition of the root ladder. On the other hand if m' were lesser than m, then m would form an inversion with m' and therefore the bar representing this inversion would be part of the route of m route. Thus, the right child of a bar x belongs to the same route as x in the root ladder.

The left child of x represents an inversion with some lesser element than m and k. Suppose this was not the case, then the left child could belong to a route greater than m, but if that were the case, this contradicts the definition of the root ladder. Thus the first element of the left child must belong to the route of some lesser element than m. Next suppose that the lesser element of the left child of x was not k. Let this element be termed k'. k' forms an inversion with the greater element of the left child of x. But since the greater element of the left child is less than m, then m would also form an inversion with k'. Thus, the bar of m and k' would be the parent of the left child, which is also a contradiction, seeing as the left child is the child of bar x. Therefore the left child of x must be a bar that it belongs to the route of a lesser element than m and its lesser element is k.

Please refer to FIG— to view an example of a root ladder with left and right children.

□

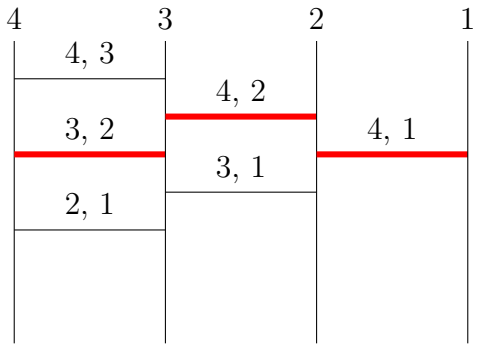


Figure 3.1: The root ladder of $(4, 3, 2, 1)$. Note that bar 4,2 is the parent of bar 3,2 and 4,1. Also note that bar 3, 2 is the the left child of 4, 2 and 4, 1 is the right child.

The second way to choose the canonical representative is to use a greedy algorithm in order to choose the best representative from the next $OptL\{\pi_N\}$. The is based on comparing the canonical representative from $OptL\{\pi_i\}$ with all the ladders in $OptL\{\pi_{i+1}\}$and choosing the ladder from $OptL\{\pi_{i+1}\}$ that had the least number of changes from the canonical ladder from $OptL\{\pi_i\}$. Each Gray Code provides a different representative. Both of the basic operations are used in order to determine the optimal canonical representative. The canonical representative from $OptL\{\pi_{i+1}\}$ is the ladder with the least number of insertions/deletions and least number of swaps when compared to the canonical representative from $OptL\{\pi_i\}$. The canonical representative for $OptL\{\pi_1\}$ is the root ladder for the idendity permutation.

In order to see a comparision for the two canonical representative selection methods for permutations of size 3 using the modified Zaks algorithm please refer to figure
_

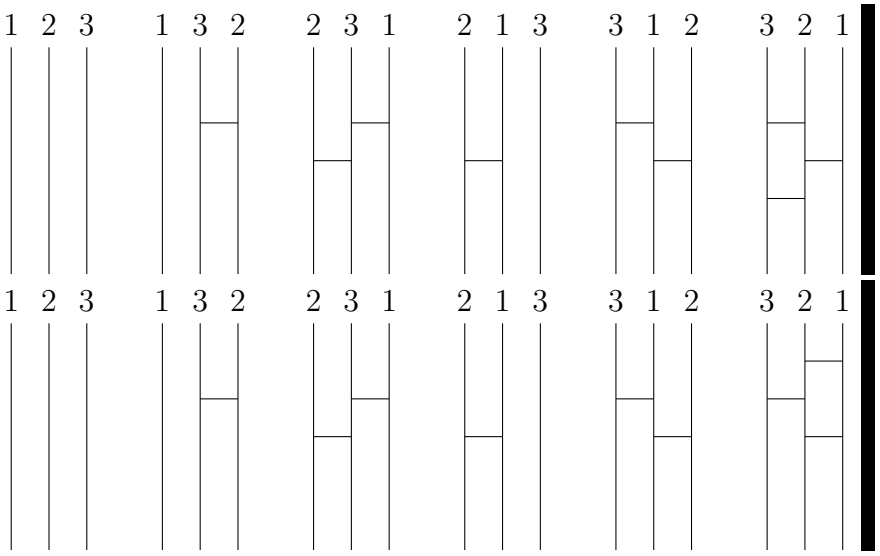


Figure 3.2: The canonical representatives from $N = 3$ for all $OptL\{\pi_3\}$ using the modified Zaks Gray Code. The top row represents the root ladder as the canonical ladder. The bottom row represents the greedy choice for the canonical ladder. Notice how the last ladder in the rows differ from each other.

3.1.2 Procedure

Thus far, the problem has been introduced and the basic operations have been defined. Recall that there are two basic operations; insertion/deletion of a bar or swapping of two bars. However, there has yet to be discussion regarding the four modified Gray Code algorithms used to generate the data. In the procedure section we look at the pseudo code for each of the algorithms and explain what each of the algorithms are doing. In each of these algorithms, only the root ladder is created. The algorithms can be modified by applying the logic to a permutation then calling the enumeration algorithm which will generate $OptL\{\pi_N\}$. This will allow the greedy approach to be applied for selecting the canonical representative.

Steinhaus-Johnson-Trotter

The principles that act as the foundation for the modified SJT algorithm following. N represents the length of the array that the ladder corresponds to. arr represents the elements from 1 to $N-1$ and is used to keep track of how many times a bar corresponding to one of their routes has been added or removed. dir represents the current direction for each element from $1 \ldots N$. The direction is switched when each element has had a bar added or removed from its route $N-1$ times going in one direction. Insert or remove a bar belonging to the route of the Nth element going from left to right or right to left. Once the first or last column has had a bar belonging to the Nth element added or removed, switch the direction for the Nth element then remove or add a bar corresponding to the $N - Rth$ element where $R > 1$. It is this stage in which $HELPERSJT$ is called. When a bar is added or removed from the $N - Rth$ element, the array at the position representing that element is increased by one. If the array at that index representing the element equals $N - R$ then reset it to zero and move onto the next element less than $N - R$. This step can be found on line(s) 3 and 10 in $HELPERSJT$. Once the $N - Rth$ element has had a bar added or removed from the first or last column, continue with $MODIFIEDSJT$ until $globalCount = n!$. The algorithm requires $O(1)$ time for the insertion or deletion of a bar. The procedure is done $n!$ times. Swapping two bars takes $O(1)$ time. The procedure is done $(n-1)!$ times.

```
1: function MODIFIEDSJT(n, ladder, arr, direction)
2:     if globalCount = n! then
3:         return
4:     end if
5:     if globalCount = 1 then
6:         ladder ← Identityladder
7:     end if
8:     dir ← direction[n − 1]
9:     for i ← 0,i < n − 1, i ← i + 1 do
10:        arr[n − 1] ← arr[n − 1] + 1
11:        if dir = left then
12:            add or remove a bar in column to the left
13:        else
14:            add or remove a bar in column to the right
15:        end if
16:        if ladder has degenerative triadic bar pattern from basicOpOne then
17:            swap the bars in the ladder
18:        end if
19:        globalCount ← globalCount + 1
20:    end for
21:    direction[n − 1] ←!direction[n − 1]
22:    HELPERSJT(n − 1, ladder, arr, direction)
23:    MODIFIEDSJT(n, ladder, arr, direction)
24: end function
```

```
1: function HELPERSJT(n, ladder, arr, direction)
2:     for i ← n − 1, i ≥ 0, i getsi − 1 do
3:         if arr[i] < i then
4:             if dir[i] = LEFT then
5:                 add or remove a bar in column to the left
6:             else
7:                 add or remove a bar in column to the right
8:             end if
9:             arr[i] ← arr[i + 1]
10:            return
11:        else
12:            arr[i] ← 0
13:            direction[i] ←!direction[i]
14:        end if
15:    end for
16: end function
```

Heap's Algorithm

```
1: function MODIFIEDHEAPS(Ladder, index, permuation)
2:     if index = 0 then
3:         return
4:     end if
5:     for i ← 0, i < index, i ← i + 1 do
         MODIFIEDHEAPS(Ladder, index − 1)
6:         if index is even then
7:             swap(perm[0], perm[index])
8:         else
9:             swap(perm[i], perm[index])
10:        end if
11:        Ladder ← Root_permuation
12:    end for
13:    if index > 1 then
14:        MODIFIEDHEAPS(Ladder, index − 1)
15:    end if
16: end function
```

The principles that act as the foundation for the modified Heap's algorithm are the following. Initially begin with the Idendity ladder, when the index is even insert swap the element in π at zero with the element and index-1. Then create the corresponding root ladder. Otherwise swap the value at the counter with the value at the index and create the corresponding root ladder. The time complexity for swapping two elements is $O(1)$. This is done $n!$ times. Add this to the cost of creating the root ladder which is n^3.

Zaks Algorithm

```
1: function MODIFIEDZAKS(permutation, ladder, suffixVector, n, index)
2:     if GlobalCount = n! then
3:         return
4:     end if
5:     endIndx ← n − 1
6:     startIndx ← n − suffixVector[index]
7:     reverse permutation's suffix starting from startIndx to endIndx
8:     ladder ← Root_permutation.
9:     index ← index + 1
10:    MODIFIEDZAKS(permutation, ladder, suffixVector, n, index + 1)
11: end function
```

```
1: function CREATESUFFIX(suffixVector, n)
2:     if n = 2 then
3:         append 2 to suffixVector
4:         return
5:     end if
       CREATESUFFIX(suffixVector, n − 1)
       temp ← suffixVector
6:     for i ← 0, i < n − 1, i ← i + 1 do
7:         append n to suffixVector
8:         append temp to suffixVector
9:     end for
10: end function
```

The principles that act as the foundation for the modified Zaks algorithm are the following. First, the suffix vector is created. The suffix vector is created with the following recurrence relation.

$$S_2 = 2$$
$$S_N = (S_{N-1}, N)^{n-1} N - 1, n > 2$$

Example 3.1.3 $N = 2$

$$S_2 = 2$$
$$(1,2),(2,1)$$

$$N = 3$$
$$S_3 = 23232$$
$$(1,2,3),(1,3,2),(2,3,1),(2,1,3),(3,1,2),(3,2,1)$$

The suffix vector represents the suffix of the permutation that is to be reversed. The

procedure begings with the Idendity permuation of size N. If N is 2 then reverse the only two elements in π. Else reverse the suffix of size $N-1$ $(N-1)!$ times, with 1 being unmoved. Then reverse the entire permutation, putting element 1 at the end of the permuation. Repeat these steps, now with element 2 as the prefix, thus leaving element 2 unmoved for the next round of $(N-1)!$ reversals. Repeat the process $N!$ which results in the reverse permuation.

Once the suffix vector is created, on each call to the main function, use the current value of the suffix vector to indicate the size of the suffix to reverse, then reverse said suffix in the permutation and create the corresponding root ladder from the resulting permutation Reversing a suffix takes $O(N)$ time. This is done $N!$ times. The cost of creating the root ladder is $O(N^3)$.

Lexicographic Algorithm

```
1: function MODIFIEDLEX(perm, ladder, n)
2:     if globalCount = n! then
3:         return
4:     end if
5:     for i ← n − 1, i > 0, i ← i − 1 do
6:         if perm[i − 1] < perm[i] then
7:             maxMin ← min value > perm[i − 1] and to the right of perm[i − 1]
8:             swap(perm[i − 1], maxMin)
9:             sort perm from index i to index n − 1 in ascending order
10:            ladder ← Root_perm
11:            MODIFIEDLEX(perm, ladder, n)
12:        end if
13:    end for
14: end function
```

The principles founding the modified lexicographic algorithm are the following. On each call to the algorithm, begin at the end of the permutation and find a decreasing subsequence going right to left. Once found, get the index of the lesser element. Then go to the right of the lesser element and find the smallest value greater than the lesser element. Swap this element with the lesser element. Sort the section of the array that is to the right of the original index of the lesser element before it was swapped. Once sorted, create the root ladder corresponding to this new permutation. The cost of sorting a suffix is $NlogN$ and is done $N!$ times. The cost of creating the root ladder is N^3.

Algorithmic Analysis

It should be noted that only the modified SJT algorithm has been customized for ladder lotteries. The rest of the algorithms require a permutation in order to generate the corresponding root ladder from the permutation. More work needs to be done in order to create the proper modified Gray Code algorithms customized for ladder-lotteries. Nonetheless, the above algorithms allow for analysis of the two basic operations which in turn can lead to determining which Gray Code is best for enumerating the canonical representative. These results will be discussed in the next section.

3.1.3 Results

The results for the two basic operations are listed in the tables below. There are two separate tables, one for the root ladder as the canonical representative, the other for the optimal ladder as the canonical representative. The results for $N = 6$ are listed. The average number of swaps and the average number of insertions or deletions are recorded for each algorithm.

Root Ladder			
Gray Code	Operatin One	Operation Two	N
SJT	1.000000	0.200000	3
HEAPS	1.800000	0.000000	3
ZAKS	1.8000000	0.2000000	3
LEX	1.400000	0.2000000	3
SJT	1.000000	0.200000	3
HEAPS	1.800000	0.000000	3
ZAKS	1.8000000	0.2000000	3
LEX	1.400000	0.2000000	3
SJT	1.000000	0.200000	3
HEAPS	1.800000	0.000000	3
ZAKS	1.8000000	0.2000000	3
LEX	1.400000	0.2000000	3
SJT	1.000000	0.200000	3
HEAPS	1.800000	0.000000	3
ZAKS	1.8000000	0.2000000	3
LEX	1.400000	0.2000000	3

Chapter 4

Evaluation

Chapter 5

Summary and Future Work

Conclude your thesis with a re-cap of your major results and contributions. Then outline directions for further research and remaining open problems.