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 listing problem. This problem asks, given all permutations of size N, is there an algorithm to list 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 two such algorithms.

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 prob-

Buddha.pdf

Figure 1.1: A picture of Buddha Amithaba

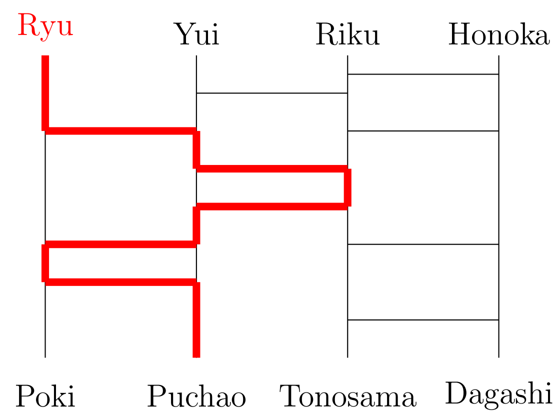


Figure 1.2: A ladder lottery where Ryu gets Puchao, Yui gets Dagashi, Riku gets Tonosama and Honoka gets Poki. You can see that Ryu's path is marked by red bars.

lems 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 and Literature Review

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**.

2.1.1 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.

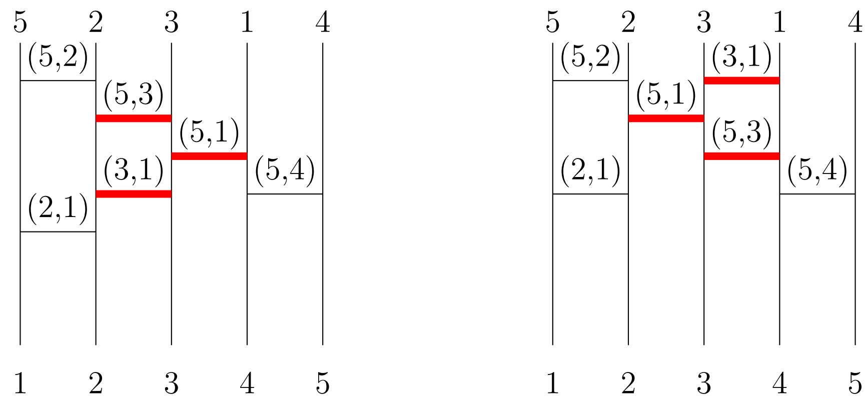


Figure 2.2: Example of a local swap operation. When a right swap operation is permformed on the left ladder, the result is the right ladder. When a left swap operation is permformed on the right ladder, the result is the left ladder.

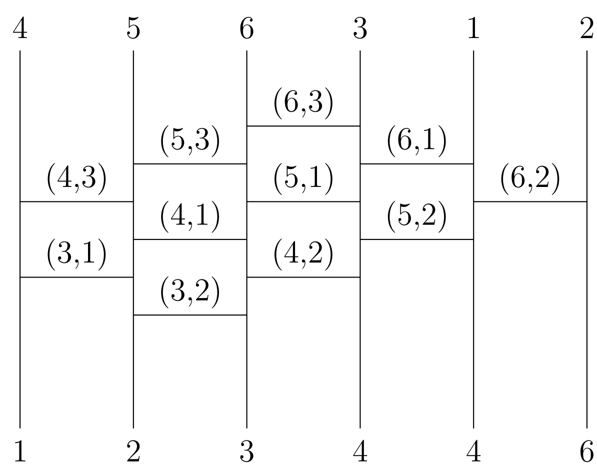


Figure 2.3: The root ladder for $OptL\{(4,5,6,3,1,2)\}$. Notice how none of the bars have undergone a right swap operation. This is clear when considering that there is no bar of a lesser element above the bar(s) of a greater element.

2.1.2 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.

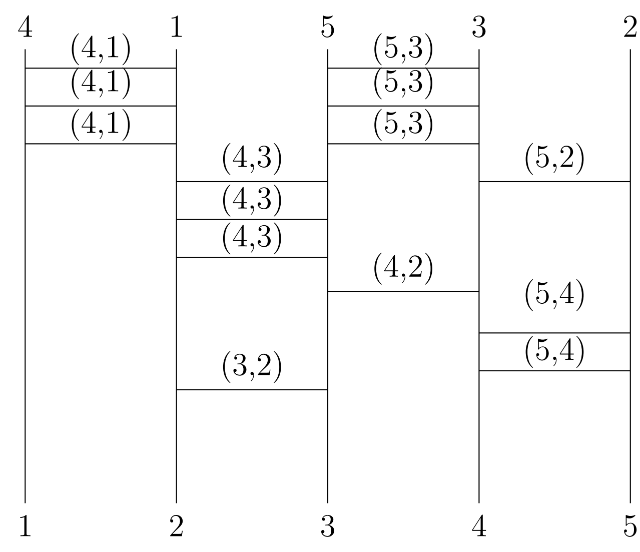


Figure 2.4: An affirmative solution to the Ladder Lottery Realization Problem given a starting perumtation $(4, 1, 5, 3, 2)$ and the multi set of bars $\{(4, 1)^3, (4, 3)^3, (4, 2)^1, (5, 4)^2, (5, 3)^3, (5, 2)^1, (3, 2)^1\}$

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 [**?**].

2.1.3 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 problem 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.

2.1.4 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 $K = [|Inv\{\pi\}| \dots N]$ [?].

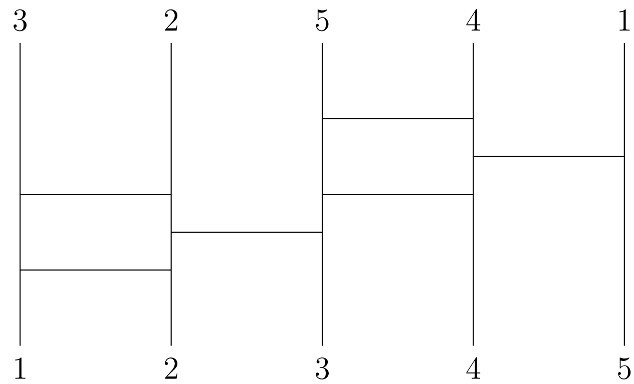
2.1.5 Coding Latter Lotteries

2.1.5.1 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 [?].

2.1.5.2 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 $C_{p_1}, C_{p_{2,\dots C_{p_N}}}$. 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 $\mathcal{O}(n^2)$.

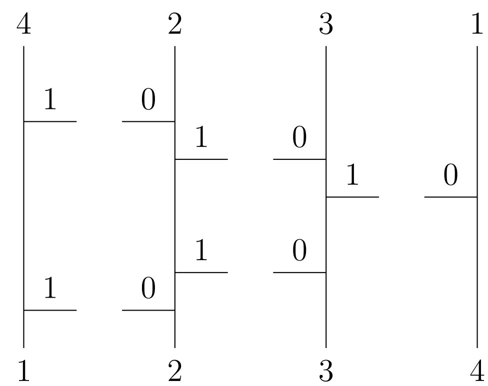


2.1.5.3 Line Based Encoding

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 $LC_L = lc_1, lc_2, \dots lc_n$. 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.

2.1.5.4 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 [**?**].

2.1.5.4.1 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}$0010.

2.1.5.4.2 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 2.1.1 *There are three possible cases for the placement of x and y in some arbitrary ladder from $OptL\{\pi\}$. 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.*

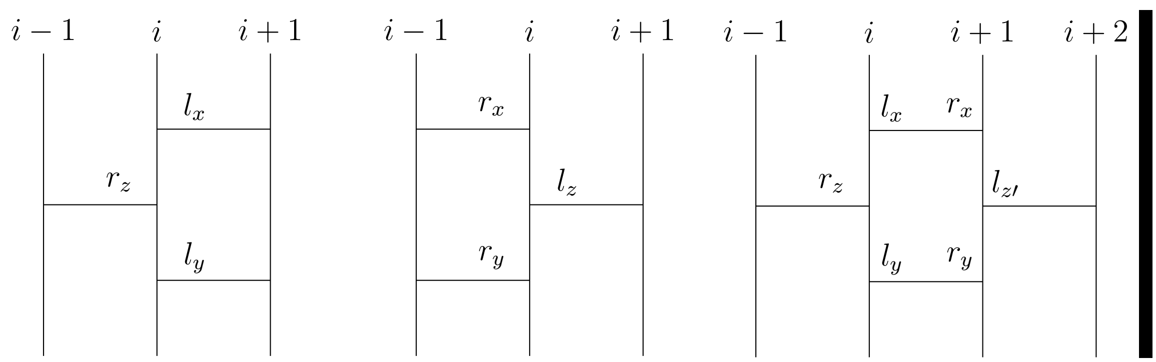


Figure 2.5: Three examples of the three cases for the placement of bars x and y in a ladder-lottery

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 $11\underline{0}010\underline{0}00\underline{0}0$.

2.1.5.4.3 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.

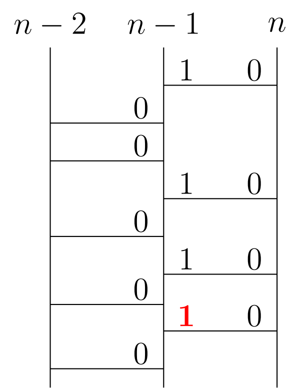


Figure 2.6: The line coding for lc_{n-1} with imrpovement three is 101110$\underline{0}$. As always, $\underline{0}$ denotes the end of the line encoding. The red, bold 1 represents the last left end point in lc_{n-1}, therefore the proceeding 0 must be included in lc_{n-1}. For every other 1 in lc_{n-1}, a 0 is omitted following said 1.

2.1.5.4.4 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 11$\underline{0}$10101$\underline{0}$0010101$\underline{0}$000. By applying imrpovement one, we get 11$\underline{0}$101011$\underline{0}$0010101$\underline{0}$. Notice how the last three 0s from LC_L were removed because they represented lc_n. By applying imrpovement two to improvememt one we get 11$\underline{0}$10011$\underline{0}$001001$\underline{0}$. Notice how the second, and eigth 1 were removed because they are implied by the successive 0s. By applying improvement three to the result of improvement two we get 11$\underline{0}$10011$\underline{0}$00101$\underline{0}$. 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 $IC_{2.9}$ = 11$\underline{0}$10011$\underline{0}$00101$\underline{0}$.

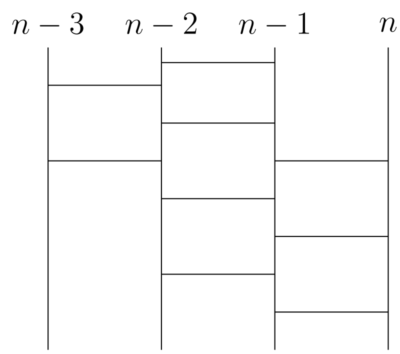


Figure 2.7: A ladder used to illustrate all three improvements IC_L. IC_L = $11\underline{0}10011\underline{0}00101\underline{0}$

2.1.6 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**.

2.1.6.1 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.

2.1.6.2 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.

2.1.6.2.1 $h < p + q$ 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.

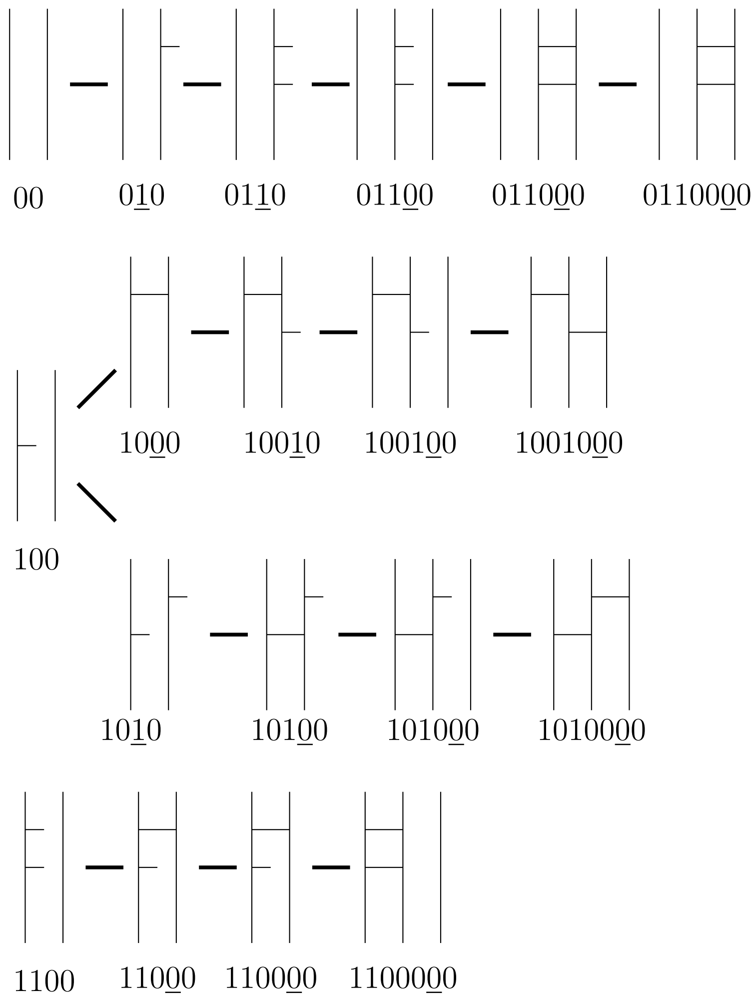


Figure 2.8: The forrest, $F_{3,2}$ where 3 is the number of lines and 2 is the number of bars. All ladders with 3 lines and 2 bars are leaf nodes of one of three trees $S_{3,2}$. The underlined bits are the inserted second last bit from the parent's line-encoding resulting in the child's line encoding

2.1.6.2.2 $n = 2$ 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.

2.1.6.2.3 ($n \geq 3$ 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 $|t(n \geq 3, h > p, q = 0)| = |t(n-1, h, 0, p)| + |t(n, h-1, p+1, q)|$.

2.1.6.2.4 $h \geq p + q$ 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 $|t(n, h \geq p+q, q > 0)| = |t(n, h-1, p+1, q)| + |t(n, h-1, p, q-1)|$ [?].

2.1.6.3 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

The Listing Problem

3.1 Introduction to the Problem

Listing problems are common problems in cambinatorics. In general, listing problems focus on enumerating the objects of a given finite set in some specific order. The listing problem in this thesis will be termed *The Canonical Ladder Listing Problem*. The problem is stated as follows: Let π_N be one of $N!$ arbitrary permutation of $[1 \dots N]$. Let *The Canonical Ladder* be a unique ladder from $OptL\{\pi_N\}$. Let $CanL\pi_N$ be the set of all canonical ladders for all $N!$ permutations of order N. Let L_i be the canonical ladder of some arbitrary permuation $OptL\{\pi_{N_i}\}$. A *change* is defined as the insertion or deletion of one or more bar(s) to get from L_i to $Li + 1$, or the relocation of one or more bars in L_i to get to L_{i+1}. The relocation of a bar is defined as moving a bar from a given row and column, to a new row and column in the ladder. The *Listing Problem* asks given all permutations of order N, is there a way to generate the canonical ladder from each $OptL\{\pi_N\}$. Furthermore, if there is a way to do so, what is the most efficient way to do so. Efficiency is defined as using minimal change to transition from L_i to L_{i+1}. For example, let $N = 4$, there are $N!$ or 24 permutations of order N. $|CanL\pi_N = 24|$; therefore there are 24 canonical ladders, one from each $OptL\{\pi_4\}$. Is there a way to generate all 24 canonical ladders for each $OptL\{\pi_N\}$? Furthermore, if there is such a way, what is the most efficient way to do so; i.e. the algorithm that requires the minimal amount of change to get from L_i to L_{i+1}. See Table – for the 24 permutations of order 4.

Each of these permutations has one or more ladders in each of their respective

1234	1243	1324	1342
1423	1432	2143	2134
2314	2341	2413	2431
3124	3142	3214	3241
3412	3421	4123	4132
4213	4231	4312	4321

Table 3.1: Table for all 4!, 24, permutations of order 4

$OptL\{\pi\}$. The canonical ladder listing problem asks, given some arbitrary $N \geq 1$, what is the most efficient way to list $CanL\{\pi_N\}$. Recall that in order to get from L_i to L_{i+1}, at least one of the two changes must be applied to L_i to get to L_{i+1}. At least one bar has to be removed/added or at least one bar has to be reolocated in L_i to get to L_{i+1}.

Theorem 3.1.1 *In order to transition from canonical ladder L_i to canonical ladder L_{i+1}, at least one bar has to be added or removed from L_i or at least one bar has to be relocated in L_i.*

Proof. We begin this proof by contradiction. Suppose L_i is some arbitrary canonical ladder for permutation of order N. Suppose that L_{i+1} is the next canonical ladder in the set of canonical ladders. Each canonical ladder represents a network of adjacent transposisitions of the corresponding permutations, π_i and π_{i+1} used to sort π_i and π_{i+1} respectively. π_i and π_{i+1} are unique. Let $Inv\pi$ be the set of all inversions is π. Let $AdjInv\pi \subset Inv\pi$ be a subset of inversions in π that are adjacent. A bar in L_i and L_{i+1} uninverts an adjacent inversion in from $AdjInv\pi_i$ and $AdjInv\pi_{i+1}$ respectively. Note, that when an adjacent inversion is uninverted, a new intermediate permuation is derived from π. Let $IntPi(\pi)$ be the permutation of intermediate permutations, beginning with π that result from performing adjacent transpositions on $AdjInv\pi$, terminating with the sorted permutation. The row in the ladder represents the order of uninverting adjacent transpositions in some intermediate permutation in $IntPi(\pi)$.

For example, row 1 in L_i represents uninverting adjacent inversions in π_i. The result of uninverting these adjacent inversions is the second intermediate permutation in $IntPi(\pi)$, π'. Row two represents uninverting the adjecent inversions in π' resulting in π'', etc. If no bars are added or removed from L_i then the number of bars in L_{i+1} is the same as in L_i. This means that the number of adjacent inversions that are uninverted in π_i is the same as in π_{i+1}. Next, suppose that no bars are relocated in L_i to get to L_{i+1}. This would mean that the same adjacent inversions in π_i exist in π_{i+1} and furthermore, the order in which these adjacent iversions were uninverted would be the same for π_i and π_{i+1}; in other words the $IntPi(\pi_i) = IntPi(\pi_{i+1})$ But this is a contradiction, seeing as pi_i and pi_{i+1} are unique. Therefore, at least one bar has to be added or removed from L_i to get to L_{i+1} or at least one bar in L_i has to be relocated to get to L_{i+1}. See fig- for an example of L_{3142} with the corresponding $IntPi(3142)$. $\square$

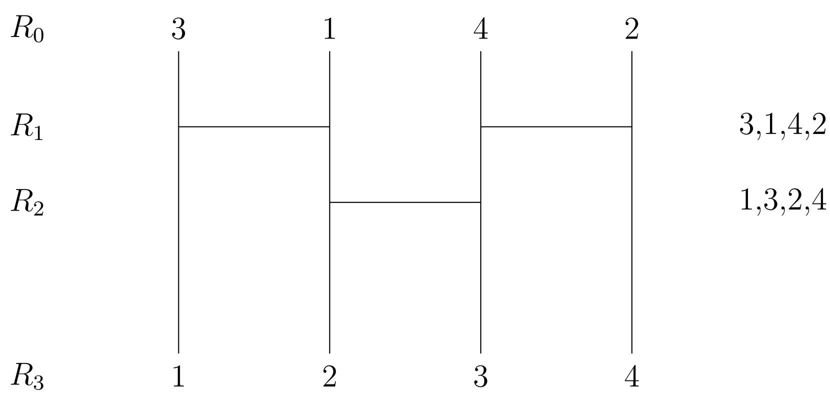


Figure 3.1: The rows are on the left of the ladder designating the order in which the adjacent inversions will be uninverted. On the right is the $IntPi(3,1,4,2)$ that results from the ladder univerting the adjacent inversions in 3,1,4,2 in the order of the rows. $IntPi(3,1,4,2) = ((3,1,4,2),(1,3,2,4),(1,2,3,4))$

In this thesis, two listing algorithms were used to generate the canonical ladders for each $OptL\{\pi_N\}$. The first of these listing algorithms is a modification of the

Steinhaus-Johnson-Trotter permutation listing algorithm. The second listing algorithm is, as far as I know, a novel algorithm. It is termed the cyclic-inversion algorithm. Both of these algorithms will be described, explained and analyzed throughout the remainder of the chapter.

Before proceeding, the justification for the canonical ladder will be presented. The canonical represeantive from $OptL\pi_N$ for $CanL\pi_N$ depends on which algorithm is being run. But in general, the canonical represeantive is the given ladder, L_i, such that minimal change is required to get from L_{i-1} to L_i.

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.

□

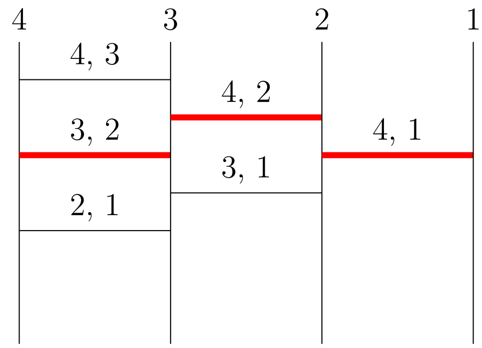


Figure 3.2: The root ladder of $(4, 3, 2, 1)$. Note that bar 4,2 is the parent of bar 3,2 and 4,1. Also note that bar 3, 2 is the the left child of 4, 2 and 4, 1 is the right child.

3.2 Procedure

Thus far, the problem has been introduced and the required terminologt has been defined. Recall that there are two changes; the insertion/deletion of bars or repositioning bars. However, there has yet to be discussion regarding the two listing algorithms. In the procedure section we look at each of the algorithms and explain what each of the algorithms are doing. The goal is to transition from L_i to L_{i+1} in $CanL\pi_N$ with minimal change, which means adding or removing the least number of bars to get from L_i to L_{i+1} or relocating the least number of bars to get from L_i to L_{i+1}.

The reason that the modified SJT and CI algorithms were chosen is because they allow for minimal change from L_i to L_{i+1}. While conducting this research, modifications to the permutation listing algorithms mentioned in chapter one were applied. Recall that these listing algorithms were Zaks, Heaps, and Lexicographic. These listing algorithms did not allow for minimal change when transitioning from L_i to L_{i+1}.

3.2.1 Steinhaus-Johnson-Trotter

Algorithm 1 Modified SJT algorithm for processing at $K = N$

function MODIFIEDSJT(N, $Ladder[2(N-1)-1][N-1]$, $Arr[N-1]$, $Direction[N]$)
    $print(Ladder)$
    if $globalCount = N!$ **then**
        return
    end if
    $dir \leftarrow direction[N]$
    $K \leftarrow N - 1$
    for $i \leftarrow 1, i < N, i \leftarrow i + 1$ **do**
        if $dir = left$ **then**
            $row \leftarrow (N) - i$
            $col \leftarrow row$
            $ladder[row][col] \leftarrow 1$
        else
            $row \leftarrow i$
            $col \leftarrow row$
            $ladder[row][col] \leftarrow 0$
        end if
        $globalCount \leftarrow globalCount + 1$
        $print(Ladder)$
    end for
    $direction[N] \leftarrow !direction[N]$
    $HELPERSJT(K, N, ladder, arr, direction)$
    $MODIFIEDSJT(N, ladder, arr, direction)$
end function

Algorithm 2 Helper SJT algorithm for processing when $2 \leq K < N$

```
function HELPERSJT(N, K = (N − 1), Ladder[2(N − 1) − 1][N − 1], Arr[N − 1], Direction[N])
    for i ← K, i ≥ 1, i ← i − 1 do
        if arr[K] < K then
            globalCount ← globalCount + 1
            if dir[K] = LEFT then
                row ← (N − 1) + (N − K) − arr[K]
                col ← (K) − arr[K]
                ladder[row][col] ← 1
            else
                row ← (N − 1) + (N − K) + arr[K] − (K − 2)
                col ← arr[K]
                ladder[row][col] ← 0
            end if
            arr[K] ← arr[K] + 1
            return
        else
            arr[K] ← 0
            direction[K] ←!direction[K]
        end ifK ← K − 1
    end for
end function
```

Let the *identity ladder* be the ladder for the sorted permutation from $[1 \ldots N]$. Let the initial conditions of the algorithm be the fallowing. The $Ladder = 2Darray$, let $n \geq 1$, let arr be set to zero for all indexes. Let dir be set to false for all indexes. The principles of the algorithm are the following, if the direction for a given route is false, then bars will be added for that given route, from right to left, bottom to top, until no more bars can be added. Let a 1 at $Ladder[row][col]$ indicate a bar has been added to the ladder at the given row and column. If the direction for a given route is true, then bars will be removed for that given route, left to right, top to bottom, until no more bars can be removed. Let a 0 at $Ladder[row][col]$ indicate a bar has been removed from the ladder at the given row and column. Let K be the value of some given route where $1 < K <= N$. Note that element one has no route. The number of bars for a given route is $1 \leq K < N$. This is because the maximum number of inversions the kth element can make is $K - 1$, therefore the kth route can have at most $N - 1$, if $K = N$, and at least 1 bar if $K = 2$. Once all the bars for the Kth route have been added or removed, the direction for the Kth route is switched, indicating that its bars will be removed if they were added, or added if they were removed. Once all the bars for the Kth route have been added or removed, the next bar of the $K - 1th$ route will be added or removed. Once this is done, the bars of route K will then be added if they were previously removed or removed if previously added. Repeat this process until all $N!$ ladders have been generated.

Theorem 3.2.1 *The number of rows required for the ladder data-structure is* $2(N - 1) - 1$ *and the number of columns required for the ladder is* $N - 1$.

Proof. The number of columns is fairly straighforward. Seeing as there are always N elements in π_N, a column represents a gap between lines in the corresponding ladder-lottery. Let $Line_i$ be a vertical line in a ladder-lottery with some element in π_N at the top of the line and the ith element in π_N be at the bottom of $Line_i$. There are N lines in the ladder-lottery, a column in the ladder data-structure simply represents a gap between two adjacent lines in the ladder lottery.

The number of rows for the ladder data-structure is calculated a follows, given π_N, the minimal number of rows required is when π_N is sorted. In this case there are zero rows because there are zero bars added to the ladder. This ladder is $L_{N_{ID}}$ and is the first ladder in $CanL\pi_N$. When a bar is added to the ladder it can be added to an already existing row or to a new row. If the current state of the ladder is $L_{N_{ID}}$ then the new bar will create the $N-1$th row in the $N-1$th column. Let the bar belong to the Nth route, then repeat adding bars for the Nth route, bottom to top left to right. Since no two bars of the Nth route can be on the same row, this will require $N-1$ rows. Note, if they were added to the same row, then the left end point of the right bar would be touching the right end point of the left bar which is disallowed. Once the bars of the Nth element are added, the bars of the $N-1th$ route will be added. The $N-1th's$ first bar will be added to the $N-2$ column, otherwise it would be directly below the first bar of the Nth route, which is a violation. Since the first bar of the $N-1$'s element is added to column $N-2$, then it must be given a new row, otherwise its right end point will be touching the left end point of the first bar of route N. The remaining $N-2$ bars of element $N-1$ will be added bottom right to top left, but none of their end points will touch the end points of element N seeing as they will always be two columns apart from any bar in $N's$ route. The same logic applies to element $N-2$, it will require one extra row for its first bar, in order not to touch the first bar of element $N-1$, but the remainder of its bars will always be two columns away from the remainder of the bars for $N-1$, etc. Therefore there are $N-2$ rows required for each element, K, $2 \leq K < N$. Note that element 1 has no bars in its route. Therefore there are $(N-1)$ rows required for element $N's$ route plus $(N-2)$ rows required for elements $2 \leq K < N$. In conclusion the number of rows required is $(N-1)+(N-2) = 2(N-1)-1$. See figure for the tree of ladders generated by modified SJT for $N = 4$ $\square$

From the above figure, it should be clear that the canonical representative from

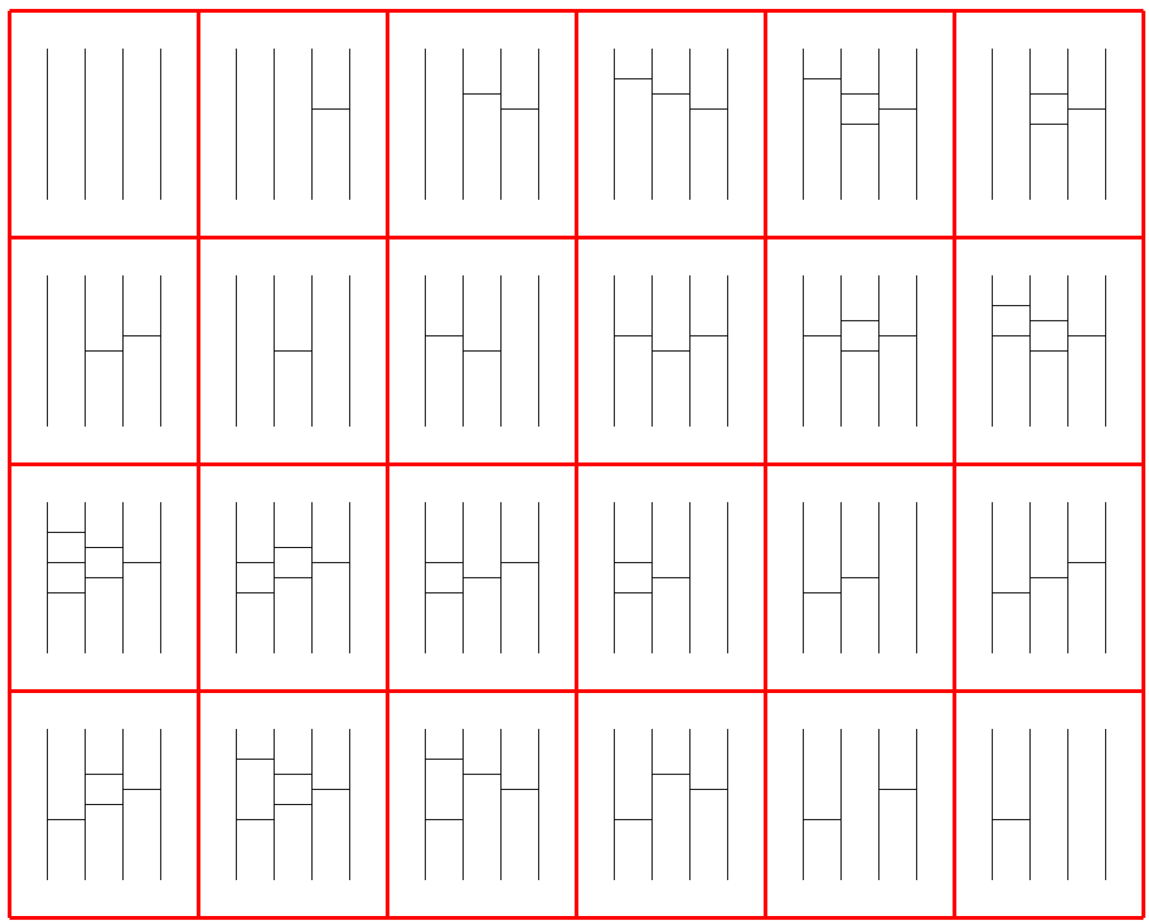


Figure 3.3: The table of $CanL\pi_4$ generated using the modified SJT algorithm. The table is to be read from top left to bottom right. Note that each ladder is the root ladder from each corresponding $OptL\pi_4$

$CanL\pi_N$ when using the modified SJT algorithm is the root ladder from each $OptL\pi_N$. Recall that the root ladder is the ladder whose bars of a lesser route have not crossed the bars of a greater route. In the case of the modified sjt algorithm, transitioning from L_i to L_{i+1} involves simply inserting a new bar or removing a bar for a given route. Let K be the current route. If a new bar being added belongs to route K, then the addition of the bar does not violate the property of the root ladder. If the new bar to be added belongs to route $K-1$, then the bar is added below $K's$ bars, still not violating the property of the root ladder. When a bar is removed, that implies it has already been added. Let L_i be a ladder whose bar is about to be removed, thus transitioning to L_{i+1}. Let L_i be a root ladder , then removing a bar from L_i cannot make L_{i+1} a non-root ladder, because removing a bar from L_i does not allow the bar of a lesser element to cross the bars of a greater element. Thus, the canonical representative for $CanL\pi_N$ is always the root ladder from each $OptL\pi_N$.

The calculations for the row and column for the bar depend on several factors. The first factor is whether the row and column is being calculated for $K = N$ or if $K < N$. If $K = N$, then the row and column are calculated using the main function, modifiedSJT. The second factor is whether a bar is being removed from the ladder or a bar is being added to the ladder. Therefore, there are eight cases to consider. The cases are the following:

Case 1: $Route = N$

Bar is being added. Row is being calculated.

Case 2: $Route = N$

Bar is being added. Column is being calculated.

Case 3: $Route = N$

Bar is being removed. Row is being calculated.

Case 4: $Route = N$

Bar is being removed. Column is being calculated.

Case 5: *Route* $< N$

Bar is being added. Row is being calculated.

Case 6: *Route* $< N$

Bar is being added. Column is being calculated.

Case 7: *Route* $< N$

Bar is being removed. Row is being calculated.

Case 8: *Route* $< N$

Bar is being removed. Column is being calculated.

When proving the above cases, keep in mind that the ladder, L, is a two dimensional array with $2(N-1)-1$ rows and $(N-1)$ columns.

Lemma 3.2.2 *Let route* $= N$. *Let* $I =$ *the current number of bars in the ladder belonging to route* N. *Assume a bar is being added. Then the row* $= (N-1) - I$.

Proof. Keeping in mind we are only dealing with root ladders, then the bars of the Nth route will be above the bars of any other route. The bars are added bottom right to top left, and no two bars of the Nth route can be on the same row, for having two bars of the same route on the same row violates the constraint that no two endpoints of two bars can be toucing. There are a total of $N-1$ rows requried for the bars of the Nth route. I is incremented for each bar that is added to the Nth route. The first bar to be added will be at row $N-1$, once it is added I is incremented by one, the second bar of the Nth route will be added to row $N-2$, which equals $N-1-I$. Then I is incremented again. This continues until all bars of the Nth route are added. Refer to figure –fig for an example of row calulation when adding a bar for the Nth route. □

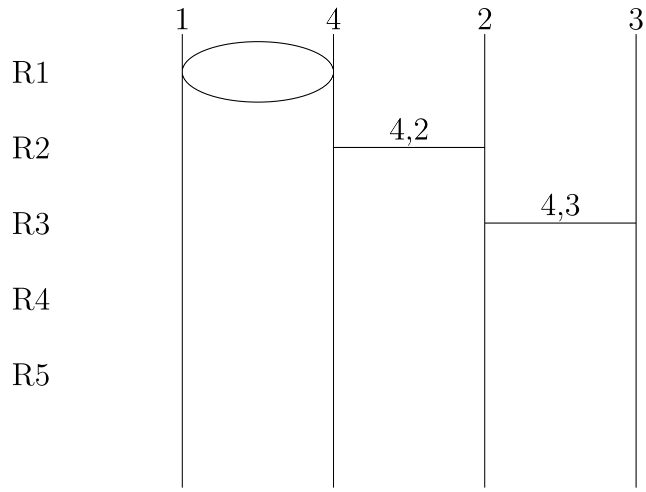


Figure 3.4: The row of the last bar to be added for element 4 is row 1. $row = 1 = 3 - 2 = (N - 1) - I$

Lemma 3.2.3 *Let route = N. Let I = the current number of bars in the ladder belonging to route N. Assume a bar is being added. Then the column = $(N - 1) - I$.*

Proof. Keeping in mind we are only dealing with root ladders, then the bars of the Nth route will be above the bars of any other route. The bars are added bottom right to top left. The ladder has a total of $N - 1$ columns, seeing as the Nth element has $N - 1$ bars, each requiring their own column. If two bars of the Nth element were in the same column, then this would violate one of two constraints. Either the two bars would be directly above/below each other, in which case the ladder would not be optimal seeing as the two elements that crossed the top bar would then cross the bottom bar, which means the ladder has an extra bar. The second case can be discredited as follows. Let the top bar belonging to route N be designated as X, let the bottom bar belonging to route N be designated as Y. Assume X and Y are in the same column. Then there is some third bar Z, not belonging to route N and not in the same column as X and Y such that Z is in the column directly to the left or right of the column of X and Y. But if that is the case, then Z is above bar Y which violates the definition of the root ladder. Therefore, every bar belonging to route N requires its own column. The first bar to be added to route N goes in the rightmost column which equals column $N-1$, then I is incremented by one. The second bar is in columb $(N-1)-1=(N-1)-I$ and I is incremented by one. The process continues until all $(N-1)$ bars of the Nth route have been added. See figure –fig for an example of column calculation. □

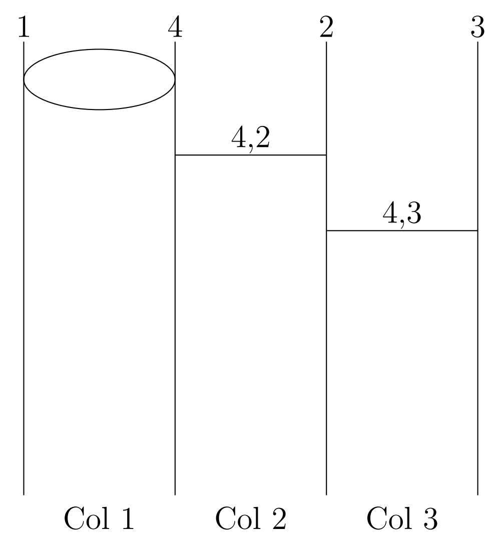


Figure 3.5: The column of the last bar to be added for element 4 is 1. $column = 1 = 3 - 2 = (N-1) - I$

Lemma 3.2.4 *Let route* $= N$*. Let* $I =$ *the current number of bars that have been removed from route* N*. Assume a bar is being removed. Then the row* $= I+1$

Proof. Keeping in mind we are dealing with root ladders and bars are removed from left to right, top to bottom, then the first bar to be removed from route N is at row one. Since no bars have been removed, I currently equals zero, thus row $1 = I+1$. Once removed, I is increased by one, indicating a bar has been removed. The next bar is at row two, which again equals $I+1$. Continue until all bars of the Nth route have been removed. See figure –fig for an example of row calculation when removing a bar for the Nth element. □

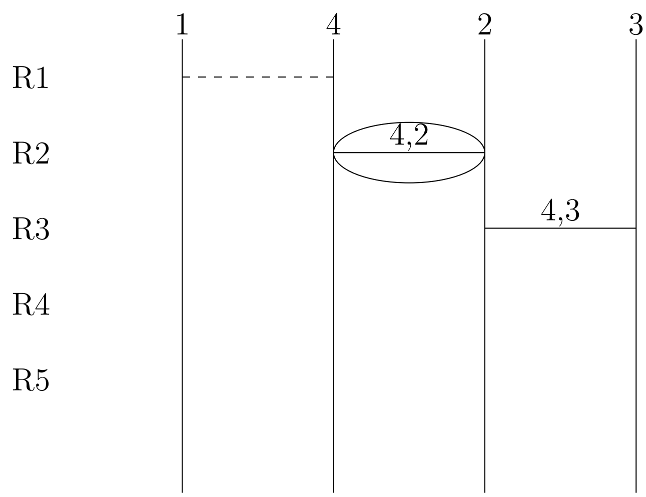


Figure 3.6: The row of the second bar to be removed from element $4's$ route is row 2. The dashed bar indicates that it has already been removed from $4's$ route. I is the number of bars currently removed from $4's$ route, which is currently 1. Therefore $row = 2 = I + 1$

Lemma 3.2.5 *Let route* $= N$*. Let* $I =$ *the current number of bars that have been removed from route* N*. Assume a bar is being removed. Then the column* $= I + 1$*.*

Proof. Keeping in mind we are dealing with root ladders and bars are removed from left to right, top to bottom, then the first bar to be removed from route N is at column one. Since no bars have been removed, I currently equals zero, thus column $1 = I + 1$. Once removed, I is increased by one, indicating a bar has been removed. The next bar is at column two, which again equals $I+1$. Continue until all bars of the Nth route have been removed. See figure –fig for an example of column calculation when removing a bar from the Nth route. □

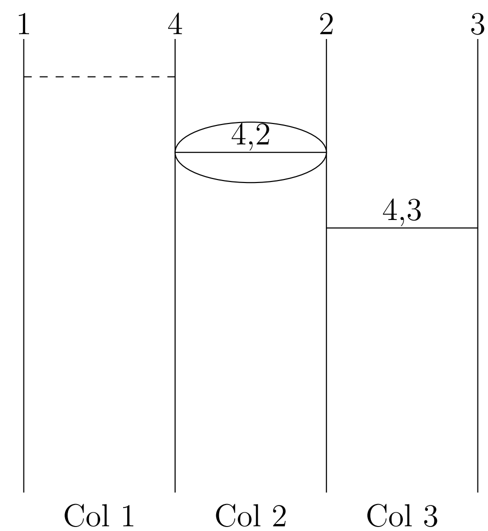


Figure 3.7: The column of the second bar to be removed from element $4's$ route is row 2. The dashed bar indicates that it has already been removed from $4's$ route. I is the number of bars currently removed from $4's$ route, which currently is 1. Therefore $column = 2 = I + 1$

Lemma 3.2.6 *Let arr be a one indexed array. Let $2 \leq K < N$ be the Kth element to have a bar added to its route. Let $arr[K]$ represent the number of bars for route K that are currently in the ladder. Let L_i be a two dimensional, one indexed array representing the current ladder. The the row for the current bar to be added for route K is $Row = (N-1) + (N-K) - arr[K]$.*

Proof. It must be noted that we are listing only root ladders. So when transitioning from L_i to L_{i+1} in $CanL\pi_N$ both are root ladders. Recall that the root ladder is the ladder such that no route of any lesser value in π has crossed the route of a greater value. With this in mind, one can say that the number of rows required for the Nth value is $N-1$ seeing as the Nth value can have at most $N-1$ bars in its route, each requiring their own row. Since bars are added right to left, bottom, up, then the first bar of route K will be added to the row just below the last bar of the previous route. The reason $N-1$ is added is because the Nth element requires $N-1$ rows in L. If K is one less than N then the first bar of K will be added one row below the last bar of N. If K is two less than N then the first bar of K will be added two rows below the last bar of N, etc. The $(N-K)$ is added because the difference between N and K is the offset of the difference in rows between the lowest/first bar of N and the lowest/first bar of K. When a bar is added to $K's$ route, the $arr[k]$ is incremented by one. This value is subtracted in order to effectively move up the ladder as bars are added to $K's$ route from bottom right to top left. See figure for an example of row calculation when adding a bar for $K < N$. □

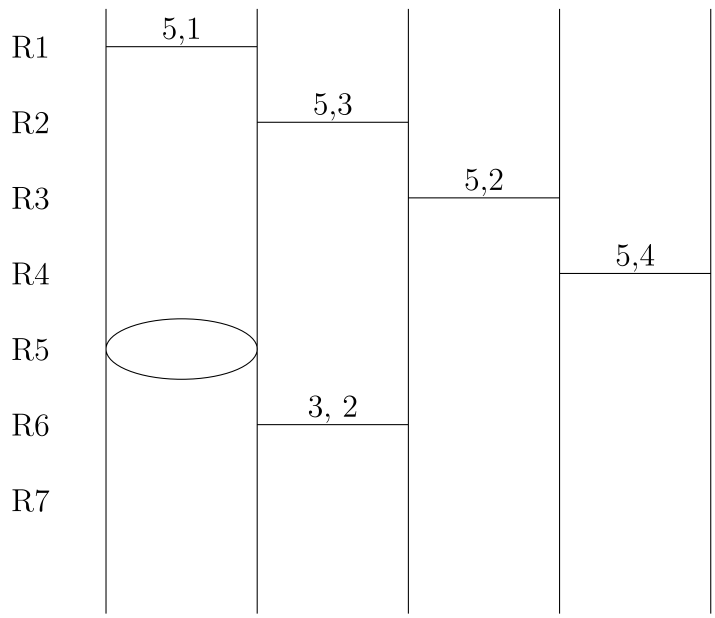


Figure 3.8: The second bar of route 3 goes will go in row 5, column 1. $5 = (5-1)+(5-3)-1 = (N-1)+(N-K)-arr[K]$.

Lemma 3.2.7 *Let arr be a one indexed array. Let $2 \leq K < N$ be the Kth element to have a bar added to its route. Let $arr[K]$ represent the number of bars for route K that are currently in L. The the column for the current bar to be added for route K is $Column = (K-1) - arr[K]$.*

Proof. The total number of bars required for route K is $K-1$, each requring their own column. The reason each bar requires its own column is the same for when the route equals N. See the proof for lemma 3.1.5. The bars are added right to left and when a bar is added $arr[K]$ is incremented by one. The initial column to add the first bar of route K is column $K-1$. This is because the first bar of the Kth route is the left child bar of the lowest bar of the $K+1th$ route. Denote the first bar to be added of the Kth route as Y and the lowest bar of the $K+1th$ route as X. X is the parent bar of Y and Y is the left child bar of X for the following reasosn. If Y was directly below X, then the ladder would have redundant bars, thus making it non-optimal. If

Y was to the right of X, then Y would either be above X, thus violating the property of the root ladder, or if Y were below X and to the right of X then Y would be part of the route for $K+1$, yet this is a contradiction seeing as we said Y belongs to $K's$ route. Therefore, Y must be in a column to the left of X. As bars are added to $K's$ route, $arr[K]$ is incremented for each bar. It is subtracted from the original column, $K-1$, effectively moving to the next column to the left in L. See figure –fig for an example of column calculation when adding a bar for $K < N$. □

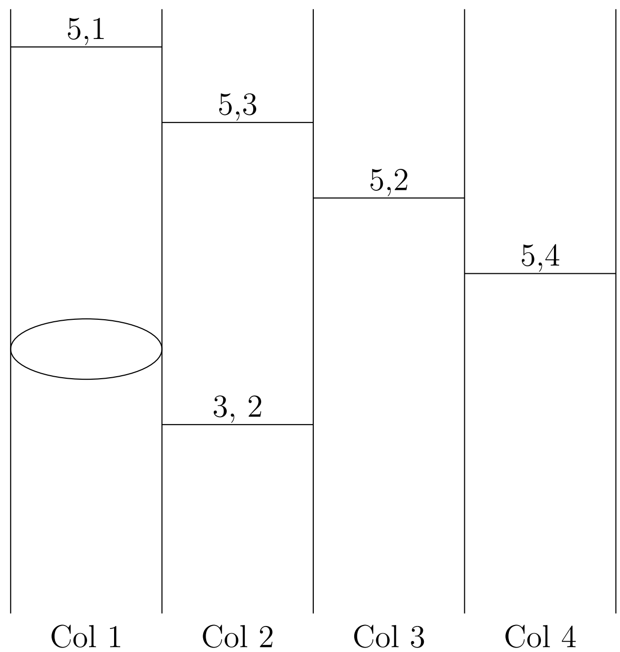


Figure 3.9: The second bar of route $K = 3$ goes will go in column 1. Since one bar has been added, $arr[3] = 1$. $col = 1 = 2 - 1 = (K-1) - arr[K]$.

Lemma 3.2.8 *Let arr be a one indexed array. Let $2 \leq K < N$ be the Kth element to have a bar removed from its route. Let $arr[K]$ represent the number of bars for route K that have currently been removed from the ladder. The the row for the current bar to be removed for route K is $Row = (N-1) + (N-K) + arr[K] - (K-2)$.*

Proof. When removing a bar the row is calculated as follows. Keeping in mind bars are removed from top to bottom, left to right. The Nth element requries the first $(N-1)$ rows. Which is why $(N-1)$ is added. The last bar to be removed of the Kth route is $(N-K)$ rows below row $(N-1)$ which is why $(N-K)$ is added. $arr[K]$ is added to effectively move down the ladder for each remaining bar of the Kth route in the ladder left to be removed. Since the first bar of the Kth route to be removed is highest up the ladder, every subsequent bar to be removed from the Kth route requires moving down the ladder from the row of first bar of the Kth route; this is accomplished by adding $array[K]$ which indicates how many bars are currently removed from the Kth route. Lastly, $(K-2)$ is subtracted in order to get to the row of the first bar of the Kth route. The difference between the row of the last bar of the Kth route and the first bar of the Kth route is $K-2$. Seeing as the Kth route has at most $K-1$ bars, each requiring their own row, then the first bar of the Kth route is $K-2$ rows higher than the last bar of the Kth route. See figure fig for an example of removing a bar. $\square$

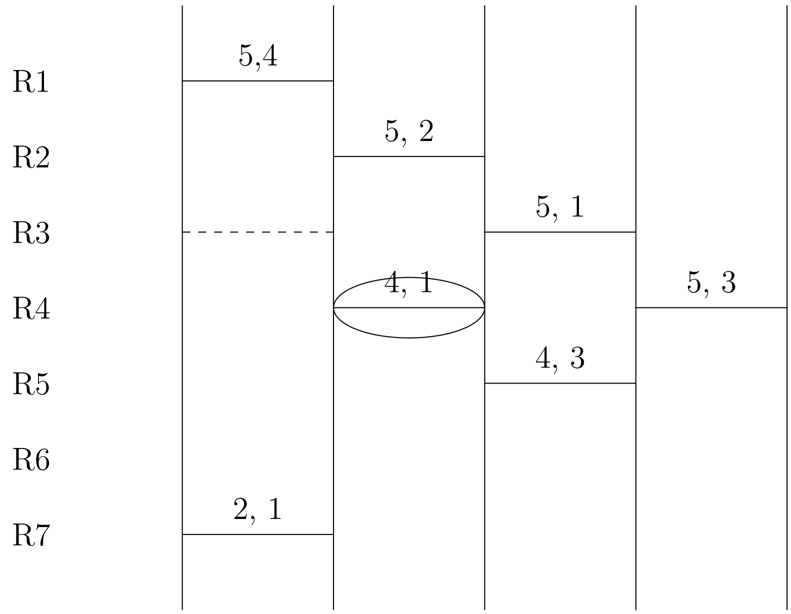


Figure 3.10: The bar to be removed for route $K = 4$ is (4, 1) which is at row 4. The dashed line indicates a bar from route 4 has already been removed. $row = 4 = (5-1)+(5-4)+1-(2) = (N-1)+(N-K)+arr[K]-(K-2)$.

Lemma 3.2.9 *Let arr be a one indexed array. Let $2 \leq K < N$ be the Kth element to have a bar removed from its route. Let $arr[K]$ represent the number of bars for route K that have currently been removed from the ladder. Then the column for the current bar to be removed for route K is $Column = arr[K] + 1$.*

Proof. The bars are removed left to right. The first bar to be removed is the leftmost bar belonging to route K which is always at column 1. This is because the number of columns required for the $K-1$ bars is $K-1$, terminating at column number $K-1$. Thus, the first bar to be removed must always be at column 1 and the last bar to be removed is at column $K-1$. $arr[K]$ is incremented for each bar removed from the route of K. □

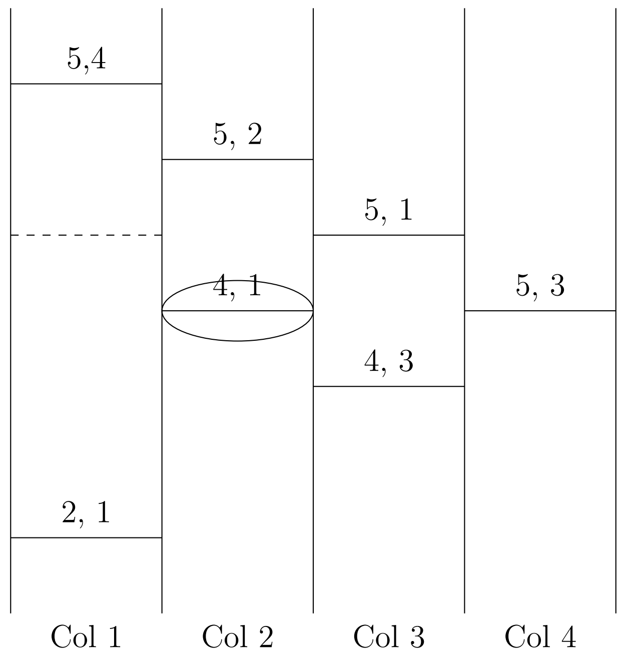


Figure 3.11: The bar to be removed for route $K = 4$ is (4, 1) which is at column 2. The dashed line indicates a bar from route 4 has already been removed. Since one bar from routr 4 has been removed, $arr[4] = 1$. $column = 2 = 1 + 1 = arr[K] + 1$.

3.2.2 Cyclic Inversion

Algorithm 3 First part of the algorithm Cyclic Inversion

```
function CyclicInversion(Ladder[2(N − 1) − 1][N − 1], CurrentLimit, MaxLimit, N, K)
    if the number of bars in Ladder = CurrentLimit then
        print(Ladder)
        return
    end if
    if CurrentLimit > MaxLimit then
        return
    end if
    if K = N then
        M ← 0
        Row ← K − 1
        Col ← K − 1
        NumBars ← current number of bars in Ladder
        while NumBars < CurrentLimit AND M < K − 1 do
            Ladder[Row][Col] ← 1
            Row ← row − 1
            Col ← col − 1
            M ← M + 1
            NumBars ← NumBars + 1
        end while
        if NumBars = CurrentLimir then
            PrintLadder(Ladder)
        end if
        remove upper leftmost bar belonging to K′s route.
        return
```

Algorithm 4 Cyclic Inversion Continued

```
    else
        count ← 0
        for I ← 0, I < K, I ← I + 1 do
            if the number of bars in Ladder = CurrentLimit then
                break
            end if
            if I = 0 then
                CyclicInversion(Ladder, CurrentLimit, MaxLimit, N, K + 1)
            else
                Row ← (N − 1) + (N − K) − count
                Column ← (K − 1) − arr[K]
                Ladder[Row][Col] ← 1
                count ← count + 1
                CyclicInversion(Ladder, CurrentLimit, MaxLimit, N, K + 1)
            end if
        end for
        remove all bars from K′s route.
    end if
end function
```

Algorithm 5 Driver for the Cyclic Inversion Algorithm

```
function CYLCLIC INVERSION DRIVER(Ladder[2(N − 1) − 1][N − 1], N)
    MaxLimit ← (N(N − 1))/2
    K ← 2
    for I ← 0, I <= MaxLimit, I ← I + 1 do
        CyclicInversion(Ladder, CurrentLimit ← I, MaxLimit, N, K)
    end for
end function
```

The initial conditions for the algorithm are the following. Let $Ladder$ be initialized as a two dimensional array with $2(N-1)-1$ rows and $(N-1)$ columns. Let N be initialized to the maximal element in π_N. Let K be initialized to 2. Let the $MaxLimit$ be initialized to $(N(N-1))/2$.Let the $CurrentLimit$ be initialized to zero. The way the algorithm works is the following. The $CurrentLimit$ represents the number of bars to be inserted into $Ladder$. Once all ladders with $CurrentLimit$ bars have been created, the $CurrentLimit$ is increased by one and the algorithm repeats until $CurrenLimit > MaxLimit$. This creates all ladders in $CanL\pi_N$. The ladders are generated as a forest structure, with each value of $CurrentLimit$ creating its own tree of ladders. See figure –fig for the forest of ladders for $N = 4$. The forest of ladders is all the ladders in $CanL\pi_N$. On each recursive call to the function, K is increased by one until $K = N$. When $K = N$ all the remaining bars that need to be added to the ladder are added to $K = N's$ route. Then the bars of $K = N's$ route are removed and relocated to the bars of $K - 1's$ route. This process repeats itself until all the combinations of bars for the $CurrentLimit$ are inserted. Each combination of bars into the $Ladder$ data structure creates a unique ladder from each $OptL\pi_N$, thus adding one more ladder to $CanL\pi_N$. Once complete, the tree of ladders terminates, and the $CurrentLimit$ increases, thus creating a new tree in the forest for $CanL\pi_N$.

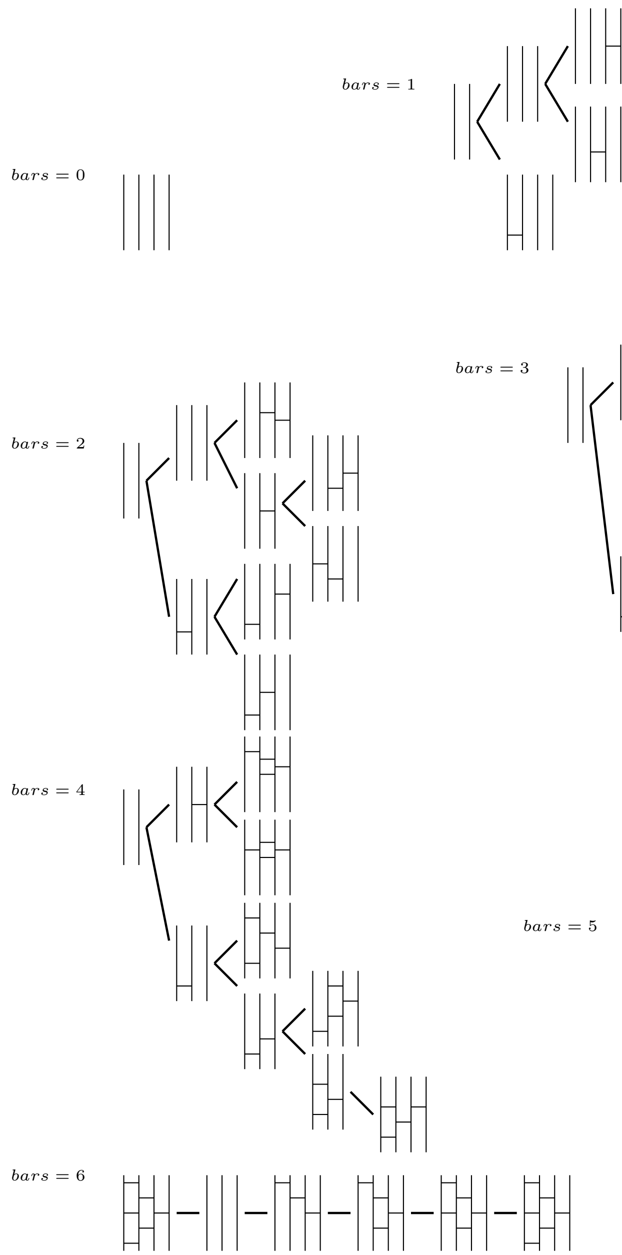


Figure 3.12: The forest for all ladders in $CanL\{\pi_4\}$ generated by the Cyclic Inversion Algorithm. The first tree has all ladders with zero bars, the second tree has all ladders with 1 bar, etc.

It has been stated that the forest created by the Cyclic Inversion algorithm generates $CanL\{\pi_N\}$. This claim has yet to have been proven, so the following lemma will prove this claim.

Lemma 3.2.10 *The forest created by the Cyclic Inversion algorithm generates* $CanL\{\pi_N\}$ $\blacksquare$

Proof. The proof is done by way of contradiction. Suppose that the cyclic inversion algorithm generated $N!$ ladders and did not generate $CanL\{\pi_N\}$, then there would be two or more ladders generated by the cyclic inversion algorithm which belonged to the same $OptL\{\pi_N\}$. If that were the case, then at least one of these ladders would not be the root ladder from the $OptL\{\pi_N\}$. However, it was already stated that the canonical representative for $CanL\{\pi_N\}$ was the root ladder from each $OptL\{\pi_N\}$, which leads to a contradiction. Therefore each ladder generated by the cyclic inversion algorithm is part of $CanL\{\pi_N\}$. □

For each tree in the forest, the algorithm effectively relocates one or more bars from the Kth route to the the $K-1th$ route until the $K-1th$ route has either no more space left for bars, i.e. the $K-1th$ route has $K-2$ bars in the ladder or the number of bars in the ladder is equal to the current limit. This is what is happening when a bar is added to some route; it only gets added once the bar of a greater route has been removed unless the route is $K=N$. When $K=N$, bars are continuously added to the $K=Nth$ route until no more bars can be added for $K=N$ or the number of bars is equal to the current limit. For example, suppose $N=5$ and the current limit for the number of bars is three, then when $K=N$, the first ladder for this forest will be the ladder in which all three bars belong to the fifth route. Seeing as element 5 has at most four bars in the ladder. However, if the current limit is 6, then the first ladder in the forest will have all the bars belonging to the fifth root as well as the first two bars belonging to the fourth root, seeing as the $5th$ element has at most 4 bars, yet the given current limit for the number of bars is six. Once all the bars for the $K=Nth$ route have been added, the upper leftmost bar of the $K=Nth$

route is relocated to the $N-1th$ route, this process continues until the $N-1th$ route has had all of its bars added. Upon completion, all the bars are removed from the $N-1th$ route, and the first bar of the $N-2th$ route is added. Then the algorithm repeats itself by adding all the bars to the $K = Nth$ route until the current limit is reached or the Nth route has $N-1$ bars added to the ladder. Again, the upper left most bar of the $K = Nth$ route is relocated to the $N-1th$ route; this continues until the number of bars added is equal to the current limit or the $N-1th$ route has $N-2$ bars added, keeping in mind that the $N-2th$ route now has a bar in the ladder. Once complete, all the bars of the $N-1th$ route are removed, and the next bar of the $N-2th$ route is added if possible or all the bars of the $N-2th$ route are removed, and the first bar of the $N-3rd$ route is added. The forest terminates when the number of bars in the ladder is equal to the current limit and each bar in the ladder belongs to the smallest route(s). For example, if $N = 4$ and the current limit for the number of bars is three, then the ladder which terminates the forest for $CurrentLimit = 3$ is the ladder with one bar belonging to route 2 and two bars belonging to route 3. It must be noted that the row and column calculation for the insertion of a bar is the same as the SJT algorithm which is why the proofs for the row and column calculation are not provided for the Cyclic Inversion algorithm.

3.3 Results

In the results section, the runtimes of the two algorithms will be provided. The run times are done without printing the ladders. When the ladders are printed, the runtime increases by a substantial amount. The runtime for each algorithm for $N = 10$ will be provided in a table. In the analysis section, the table will be further analyzed along with the time and space complexity for each algorithm.

Runtimes for generating $CanL\{\pi_N\}$ in seconds		
N value	Cyclic Inversion	Modified SJT
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000
4	0.000000	0.000000
5	0.000000	0.000000
6	0.000000	0.000000
7	0.000000	0.000000
8	0.000000	0.000000
9	0.093750	0.000000
10	0.968750	0.031250
11	12.718750	0.250000
12	174.312500	2.781250

Table 3.2: The table with the runtimes for listing $CanL\{\pi_N\}$ using the Cyclic Inversion Algorithm and Modified SJT Algorithm.

3.4 Analysis

3.4.1 Introduction

From looking at the table in the results section, it is cear that the modified SJT algorithm performs better than the Cyclic Inversion algorithm. The reson(s) for this disparity in performance will be analyzed. Following this analysis, areas of application and practical relavence for the Listing Problem will be discussed along with concluding remarks.

3.4.2 Performane Analysis

As $N \geq 9$ there is a noticeable difference between the runtimes of the two algorithms by a sizable order of magnitude. Cleary the modified SJT algorithm performs better than the Cyclic Inversion algorithm. The reason(s) for this improved performance are the following. Firstly, the time complexity of the two algorithms are different. The time complexity for the modified SJT algorithm is $(N!)N$. The time will be proven in the following lemma.

Lemma 3.4.1 *The time complexity for the modified SJT algorithm is* $O((N!)N)$

Proof. The $N!$ factor is fairly straightforward, the algorithm creates all $N!$ ladders in $CanL\{\pi_N\}$ which accounts for the $N!$ factor. The N factor is a result of the second for loop found in the algorithm. The first for loop fount in the modified sjt function runs $(N-1)$ times each time the modified SJT function is called, however on each iteration of this for loop a ladder is listed, therefore the runtime of this for loop is accounted for by the $N!$ factor. However, the second for loop in the helper SJT function runs at worst, $N-1$ times before listing a ladder. This worst case is when the $K = 2$ route needs to have a bar inserted or removed. Therefore, this second for-loop accounts for the N factor in the time complexity. Thus, the time complexity of the modified SJT algorithm is $O((N!)N)$. □

On the other hand, the time complexity of the Cyclic Inversion algoirthm is $O((N!)N^2)$. The time complexity for the Cyclic Inversion Algorithm will be proven in the following lemma.

Lemma 3.4.2 *The time complexity for the Cyclic Inversion algorithm is* $O((N!)N^2)$

Proof. The $N!$ factor is fairly straightforward, the algorithm creates all $N!$ ladders in $CanL\{\pi_N\}$ which accounts for the $N!$ factor. The N^2 factor is a result of the for loop that is executed when $2 \leq K < N$. This for loop runs from 1 to K for each value of K. Thus, the for loop is executed $1 + 2 + 3 + 4, ... + N - 1$ times. This summation is equal to $((N-1)N-2)/2$ which is reduced to N^2. Therefore the for-loop when $2 \leq K < N$ accounts for the N^2 factor. □

3.4.3 Application(s)

The applications for generating $CanL\{\pi_N\}$ are currently unknown insofar as this problem has yet to be solved to my knowledge. However, if I am to be granted some speculation, I could provide some hypothetical scenarios in which listing $CanL\{\pi_N\}$ could be of interest. The first hypothetical application could be to model an *oblivious sorting system* for $N!$ permutations. An oblivious sorting system is a system such that the sorting operations are done irrespective of the data being passed to the system. Recall that a bar in a ladder simply swaps two adjacent elements in a permutation. Due to the static nature of each ladder, the swap operation resulting from two elements in a permutation crossing a bar is unchanging. Seeing as each ladder in $CanL\{\pi_N\}$ sorts the corresponding permutation of order N, one can implement all of $CanL\{\pi_N\}$ for some arbitrary N value and then pass each permutation of order N through its respective ladder from $CanL\{pi_N\}$ thus resulting in each permutation being ordered. The ladders from $CanL\{\pi_N\}$ only need to be generated once and saved. Once this is done a permutation can be passed to the correct ladder and it can be sorted by having each of its elements pass through the ladder.

Chapter 4

The Minimum Height Problem

4.1 Introduction To The Problem

The Minimum Height Problem an optimization problem relating to ladder lotteries. Let the *height* of a ladder be the number of rows that a ladder has. The Minimum Height Problem asks, given $OptL\{\pi\}$, what ladders in the set have the shortest height? Let $MinL\{\pi\} \subseteq OptL\{\pi\}$ such that the ladders in $MinL\{\pi\}$ are the shortest ladders from $OptL\{\pi\}$. Let *a minimal ladder* be a ladder from $MinL\{\pi\}$. Given a permutation π, is there an algorithm for generating a minimal ladder from $MinL\{\pi\}$?

Some tangential questions that result from this problem are the following. Let $MinL\{\pi_N\}$ be the set of all $MinL\{\pi\}$ for each permutation of order N. Recall that $OptL\{\pi_N\}$ is the set of all $OptL\{\pi\}$ of order N. Thus, $MinL\{\pi_N\} \subseteq OptL\{\pi_N\}$. The first tangential question is, what are the upper and lower bounds for a minimal ladder in $MinL\{\pi_N\}$? Let *ladders of order* N pertain to ladders derived from some π with N elements. The second tangential question is what ladders of order N have a height of zero or one? Thirdly, is $|MinL\{\pi\}| = 1$, or in other words is there only one ladder from $OptL\{\pi\}$ with a minimal height?

Firstly I will address the tangential questions in the introduction. Following the tangential questions, I will provide a heuristic algorithm for generating one ladder from $MinL\{\pi\}$ in the procedures section. In the results section I will provide a table with the heights of the ladder from the heuristic algorithm in comparison to the heights of the ladders in $MinL\{\pi\}$. Finally, in the analysis section there will be a discussion about the efficacy of the heuristic algorithm along with some applications of the algorithm.

4.1.1 Upper and Lower Bounds of the heights of the Ladders in each $MinL\{\pi_N\}$

In order to address the question as to what the upper and lower bounds for each $MinL\{\pi_N\}$ some points of clarification need to be addressed. It must be noted that each $MinL\{\pi_N\} \subseteq$ of each corresponding $OptL\{\pi_N\}$. For example, let $N = 4$, there are 24 or 4! $OptL\{\pi\}$ in $OptL\{\pi_4\}$, which is to say each permutation of order 4 has its own $OptL\{\pi\}$. Each $MinL\{\pi_4\}$ is a subset of one of the 24 $OptL\{\pi_4\}$. We are going to determine what the upper and lower bounds for the heights of the ladders in $MinL\{\pi_N\}$ are; not the upper and lower bounds for the heights of the ladders in $OptL\{\pi_N\}$. Although the lower bound for the height of a ladder in $MinL\{\pi_N\}$ will also be the lower bound for the height of a ladder in $OptL\{\pi_N\}$ seeing as the ladder from $MinL\{\pi_N\}$ that has the lower bound for its height will be the shortest ladder from all $OptL\{\pi_N\}$.

Lemma 4.1.1 *The lower bound for the height of a ladder $MinL\{\pi_N\}$ is zero*

Proof. If π_N is the sorted permutation of order N then there are no bars in its ladder. Recall that a bar swaps an adjacent inversion in π. Seeing as there are no adjacent inversions in the sorted permutation of order N, then there are no bars that need to be added to its corresponding ladder. Since a ladder with no bars requires no rows, then the lower bound for the height of a ladder from $MinL\pi_N$ is zero. This is the ladder belonging to $OptL\{\pi_{ID_N}\}$. □

The upper bound for the heights of the ladders in $MinL\{\pi_N\}$ is more difficult to prove than the lower bound. The lower bound is unique seeing as there is only one ladder of order N with zero bars. With the upper bound however, it has yet to be shown if there is an upper bound for $MinL\{\pi_N\}$. Before proving the upper bound for $MinL\{\pi_N\}$ it must be shown how to derive the ladder with minimal height from the root ladder of the reverse permutation of order N. Once we have established how to derive the ladder with minimal height from the root ladder of the reverse permutation of order N, it will be relatively easy to prove the upper bound for $MinL\{\pi_N\}$.

Let $Degen_{\pi_N}$ be the reverse permutation of order N. Let $MinL_{Degen_{\pi_N}}$ be a ladder with the shortest height for $Degen_{\pi_N}$. Let R_{Degen} be the root ladder from $OptL_{\{}Degen_{\pi_N}\}$. Recall that the root ladder is the ladder such that no bar of a lesser element has crossed the route of a greater element. R_{degen} requires $2(N-1)-1$ rows. The Nth element requires $N-1$ rows seeing as each of the bars in its route cannot be on the same row as any other bar in the same route. The route of the Nth element spans from the first column to the $N-1th$ column. The $N-1th$ element requires $N-2$ bars. Seeing as the $N-1th$ element is directly to the right of the Nth element in $Degen_{\pi_N}$ and it requires $N-2$ bars in R_{Degen}; its firt bar in R_{Degen} will be in the first column and its last bar will be in the $N-2th$ column. Since the endpoints of no two bars can be touching, the last bar of the $N-1th$ route will be one row below the last bar of the Nth route. The same pattern applies to the $N-2th$ element in relation to the $N-1th$ element and so on. Since all the bars of a lesser route in R_{Degen} must be below the route of any greater element, this means the first bar of any route will begin at column one in the ladder. Since each bar of the $N-Kth$, $0 \leq K < (N-1)$, element requires $N-K-1$ bars in its route, the route will span from column one to column $N-K-1$; each bar of the route cannot share a row with any other bar in the route. Yet since the last bar of the previous element's route is at the currently lowest row in the ladder, a new row will need to be added to the ladder to accomodate the last bar of the current element.

In order to create a ladder with minimal height from $OptL_{\{}Degen_{\pi_N}\}$, one simply needs to take R_{Degen} and modify it. In order to modify R_{Degen} correctly, consider what happens when the bars of lesser elements are swapped above the bars of greater elements. Of course, if this is done then the ladder is no longer R_{Degen}. Nonetheless, when the $N-1th$ route is swapped above the Nth route, this frees up an extra row in the ladder for the $N-2th$ route. This is the row where the last bar of the $N-1th$ element resided before it was swapped above the Nth route. Now, the first bar of the $N-1th$ route will begin in column 2 and end at column $N-1$. Furthermore, a new row will need to be added to the top of the ladder in order to accomodate the first bar of the $N-1th$ route. Now the route of the $N-2th$ element can be raised up a row seeing as its last bar will still be in column $N-2$ and the row that was previously occupied by the last bar of the $N-1th$ element will be free. Then the $N-3$ route can be swapped above element N and begin at column 4 and span to column $N-1$. Since a new row was already added aboev route N for element $N-1$ and the first bar of element $N-1$ route began at column 2, the first bar of element $N-3$ and go in the same row as element $N-1$ seeing as the only other bar in this new row is at column 2. By swapping all the $N-Jth$, $1 \leq J < (N-1)$ and $J = 2K+1$, routes above the route of the route of the Nth element in R_{Degen}, the ladder is reconfigured to have the minimal height. This height is N because the Nth element still requires $N-1$ rows, and the $N-1th$ element will require a new row to be added above the row of the first bar of the Nth element to accomodate its first bar. Essentially, if N is even, then swap the route of each odd element in R_{Degen} above route N and keep the route of each even element below route N to create a ladder with N rows. If N is odd, then swap the route of each even element in R_{Degen} above the Nth element and keep the route of eahc odd element below the route of N to create a ladder with N rows. Please refer to figure –fig for an example of modifying $R_{5,4,3,2,1}$ to $MinL_{5,4,3,2,1}$.

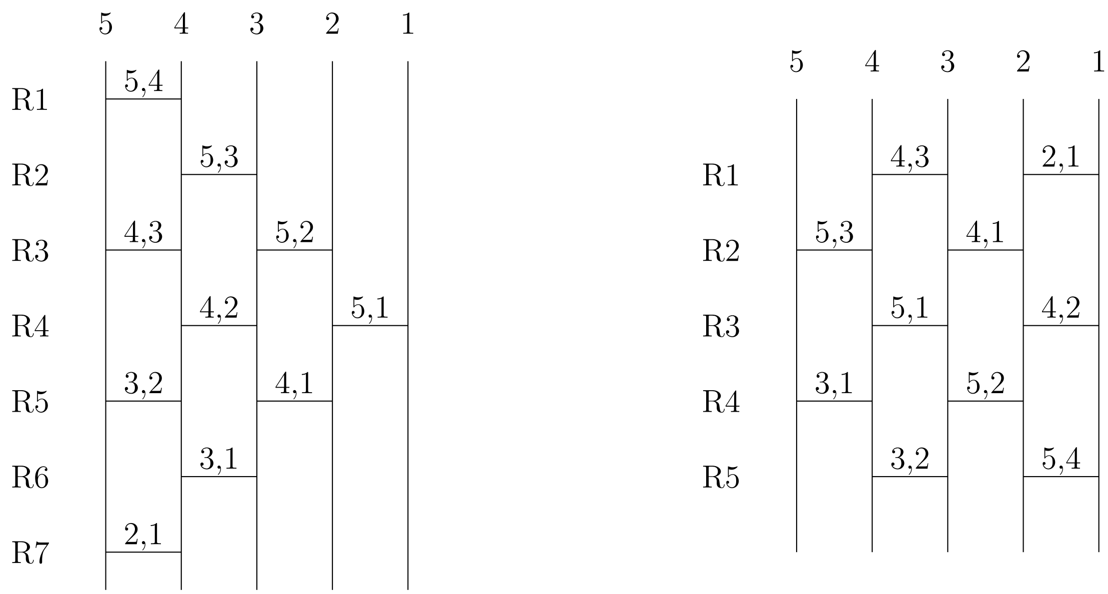


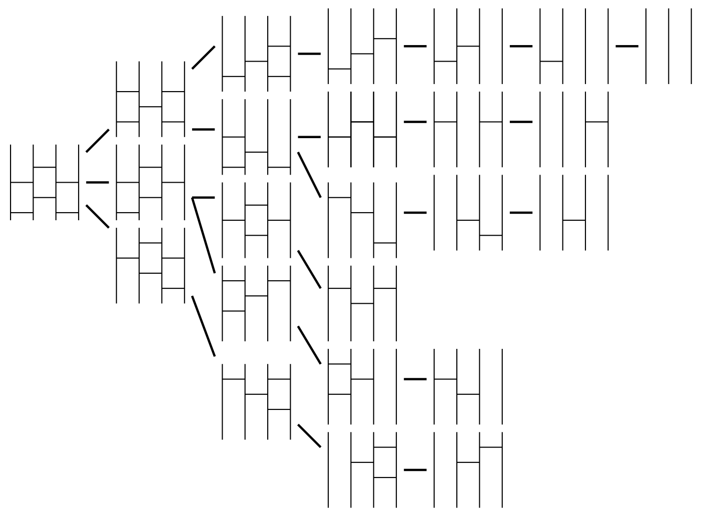
Figure 4.1: The ladder to the left is $R_{5,4,3,2,1}$. The ladder to the left is $MinL_{5,4,3,2,1}$. Note that $N = 5 = 2K+1$, thus by swapping routes 2 and 4 above route 5 whilst leaving route 3 below route 5 in $R_{5,4,3,2,1}$, we get $MinL_{5,4,3,2,1}$. The height of $MinL_{5,4,3,2,1}$ is 5. There is no way to reduce the height seeing as route 5 still needs 4 rows and route 4 needs one extra row for its first bar.

Now that $MinL_{Degen_{\pi_N}}$ has been established, we will go back to proving the upper bound for $MinL\{\pi_N\}$.

Lemma 4.1.2 *The upper bound for $MinL\{\pi_N\}$ is N.*

Proof. We shall use a proof by contradiction. Suppose that the upper bound for the height of $MinL\{\pi_N\}$ was greater than N. (It cannot be less than N because we have already demonstrated that the minimal height of the ladder for the reverse permutation is N). Let $MinL_{Degen_N}$ be the minimal ladder for the reverse permutation of order N. Refer to figure –fig for an example of $MinL_{5,4,3,2,1}$. It will be shown that for each ladder of order N can be created by deriving it from $MinL_{Degen_N}$. Recall that a bar simply univerts an inversion in a permutation. By removing bars from $MinL_{Degen_N}$, that is effectively removing inversions from $Degen_{\pi_N}$. Of course, when a bar is removed from $MinL_{Degen_N}$, the laddr ceases to be $MinL_{Degen_N}$. Let K be the number of bars in the current state of the ladder, wth $MinL_{Degen_N}$, $K = (N(N-1))/2$. Fpr each subsequent ladder, $0 \leq K < (N(N-1))/2$. Thus, to create the minimal ladders with $K = ((N(N-1))/2) - 1$ bars , simply one of the correct bars from $MinL_{Degen_N}$. Once all the minimal ladders with $K = ((N(N-1))/2) - 1$ bars have been created, simply remove the correct bar from each of these ladders with $K = (N(N-1))/2 - 1$ bars to get all minimal ladders with $K = ((N(N-1))/2) - 2$ bars. This process continues until each minimal ladder of order N has been created. Since bars are only being removed from the initial ladder which is $MinL_{Degen_N}$, no more rows will be added to the ladder. Removing a bar does not necessarily remove a row, but removing a bar definitely does not add a row to the ladder. Earlier we stated that the height of $MinL_{Degen_N}$ is N, and at the same time we stated that we could create a minimal ladder of order N by deriving it from $MinL_{Degen_N}$ through removing bars. Yet at the beginning of the proof, we supposed the upper bound was greater than N which contradicts the claim that by removing bars from $MinL_{Degen_N}$ the height of $MinL_{Degen_N}$ will not increase. Thus, the upper bound for $MinL_{\{}\pi_N\}$ is N. Please refer to figure –fig for each ladder with minimal height generated derived from $MinL_{4,3,2,1}$.

□



4.1.2 Minimal Ladders of Order N with Heights of Zero or One

There are some ladders of order N which have a height of zero or one. There is only one permutation of order N which results in a minimal ladder with a height of zero, namely the identity permutation. This point has already been proven in the lemma for the lower bound of the minimal height. What is more interesting is ladders of order N with a height of one. One may be tempted to assume that if the identity permutation results in a minimal ladder with a height of zero, then all permutations of order N with exactly one inversion result in minimal ladders with a height of one. Although this is true, it is only partially true. There are more permutations of order N with more than one inversion which result in minimal ladders with a height of one. Below will be presented one algorithm, one recurrence relation and one formula pertaining to ladders of order N with a height of one. The algrithm lists all ladders of order N with a height of one. The recurrence relation counts all ladders of order N with a height. The formula is the closed form solution to the recurrence relation. The similarities between ladders of order N with a height of one and other mathematical objects will close off the topic.

4.1.2.1 Listing Algorithm for all Ladders of Order N with a Height of One

Algorithm 6 Listing Algorithm For All Ladders of Order N with a height of 1

```
function GenHeightOne(Ladder[1][K = N − 1], Col = N − 1)
    if Col < 1 then
        return
    end if
    Ladder[1][Col] ← 1
    GENHEIGHTONE(Ladder, Col − 2)
    Ladder[1][Col] ← 0
    GENHEIGHTONE(Ladder, Col − 1)
end function
```

Let *Ladder* be a two dimensional array initialized as the identity ladder of order N. Let *Col* be initialized to $N-1$ indiciating the current column. When a 1 is inserted at $Ladder[1][Col]$ that indicates a bar has been added to row 1, Col. When a 0 is inserted at $Ladder[1][Col]$ that indicates a bar has been removed from row 1, Col. Since no two endpoints of two bars can be touching, the function moves two columns to the left on the first recursive call. This ensures that the next bar added will be two columns away from the current bar that was just added. Once the Col is less than 1 the function returns to the previous value of Col and removes the bar that was at $Ladder[1][Col]$. This now frees the column that is one away from the value of Col. Thus, the function makes a second recursive call, this time reducing Col by one. Each call to the function produces a unique ladder. To see the tree of all ladders with a height of one for $N = 5$ please refer to figure –Fig

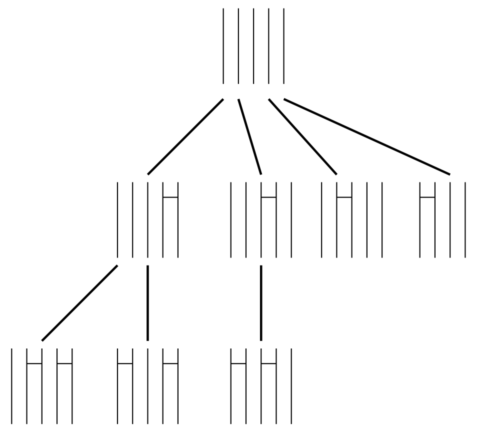


Figure 4.2: All 7 ladders of order 5 with a height of one listed by the function *GenHeightOne*

4.1.2.2 Recurrence Relation for Counting the Number of Ladders of Order N with a Height of One

Although it may seem redunant to provide a recurrence relation to count the number of ladders of order N with a height of one, seeing as there is already an algorithm for listing all ladders of order N with a height of one, one can use the recurrence relation to prove the veracity of the listing algorithm. Also, the recurrence relation of the number of ladders of order N is the same recurrence relation for other combinatorial objects such as the number of involutions in the Symmetric Group S_{N-1} or the number of permutations, π, of 1, 2, ..., n - 1 such that $max|\pi_i - i| = 1$.

Theorem 4.1.3 *The recurrence relation for the number of ladders of order N with a height of* 1 *is:*

$$\begin{cases} L(0) = 0 & N = 0 \\ L(1) = 0 & N = 1 \\ L(N) = L(N-1) + L(N-2) + 1 & N \geq 2 \end{cases}$$

Proof. We shall do a combinatoial proof to demonstrate the above theorem. Suppose we want to count all binary strings of length N such that there can be no consecutive $1s$ and there must be at least one 1 in the string. Suppose we are counting $1s$ from right to left. Suppose the first 1 in a binary string of length N is at position N, then the second 1 can appear at position $N-2$, thus we have binary strings of length N with the first 1 appearing at position N and the second one appearing at position $N-2$; let $M =$ the number of binary strings of length N such that there is a 1 at position $N-2$. Suppose a binary string of length N has a 0 at position N, then the first 1 can appear at position $N-1$ or position $N-2$. If it appears at position $N-2$ we have binary strings of length N with a 1 at position $N-2$. We already designated this number as M, so we get $2(M)$. Still supposing we are considering binary strings of length N with a 0 at position N, consider all binary strings of length $N-1$ with no consecutive $1s$. Let $K =$ the number of binary strings of length $N-1$ with no consecutive $1s$ and at least one 1. Let the first 1 in the binary string of length N appears at postion $N-1$, then we have $2(M) + K$. Still assuming a 0 at position N in binary strings of length N, if there is also a 0 at position $N-1$, then the first 1 can appear at position $N-2$. The number of binary strings of length N with a 1 at position N was designated as M. Thus we have $2(M) + M = 3M$ Yet we have already counted M under the conditions that the first 1 in binary strings of length N appears at position $N-2$. Therefore we subtract M from K thus leaving us with $J =$ the number of binary strings of length N with the first 1 appearing at position $N-1$. Now we have $2(M) + J$. Then consider all binary strings of length N such that from positions $1 \ldots N-1$ there are only $0s$. Therefore there must be a 1 at position N seeing as we are considering all binary strings of length N with at least one 1. Since only one such binary string of length N exists we simply add one. Thus we get $2(M)+(K-M)+1 = 2(M)+J+1 =$ the number of binary strings of length N with at least one 1 and no consecutive $1s$.

Now consder a ladder, L, with $N+1$ lines. The number of columns in L is N. The stipulation of L is that L has a height of one. Note that the end points of no two bars can be touching which is to say that there can be no adjacent bars on the same row. For example, if there is a bar at row 1, column N then the next consecutive bar in row 1 can appear at most at column $N-2$. Knowing this, we can easily see how this scenario models all binary strings of length N with no consecutive $1s$ and having at least one 1. Let a bar in L be represented as a 1 in a binary string of length N. Knowing that a ladder with zero bars has a height of zero, it must be the case that L has at least one bar. Thus we get the same formula for the number of ladders of order $N+1$ where M is the number of ladders with a bar appearing in column $N-2$, $(K-M) = J$ being the number of ladders with the first bar in column $N-1$ minus all ladders with the first bar appearing at column $N-2$. Lastly is the $+1$ for all ladders of order $N+1$ where the only bar appears at column N. See figure –Fig for the mapping of binary strings of length $N = 5$ with no consecutive $1s$ and at least one 1 to ladders of order 6 with a height of one.

□

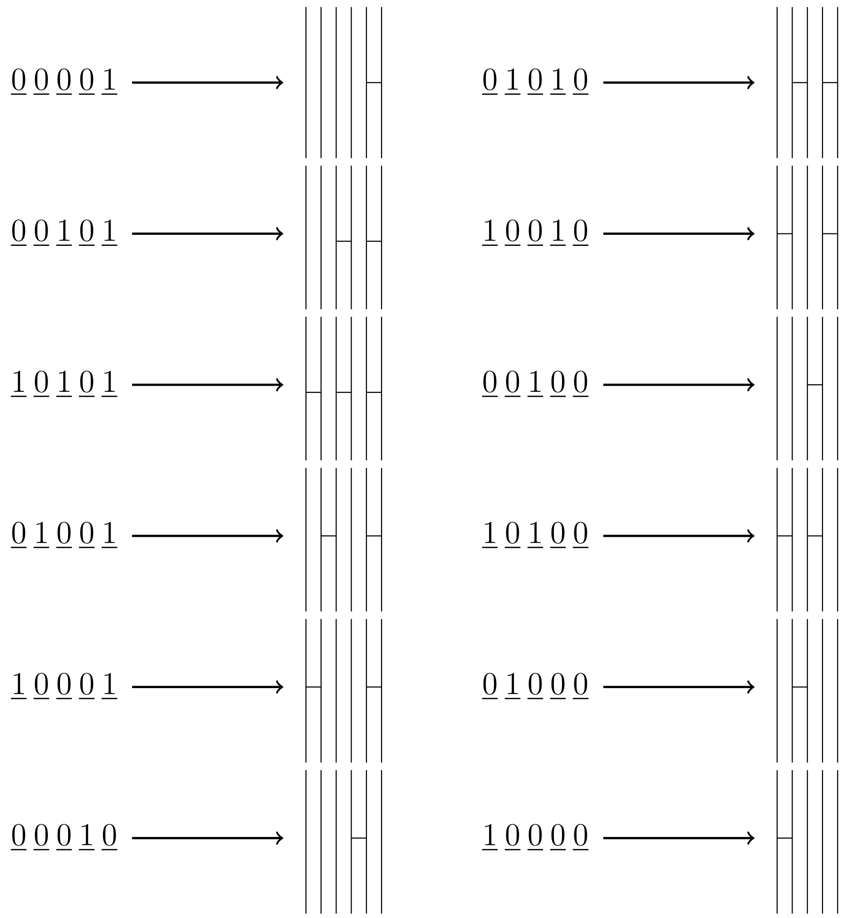
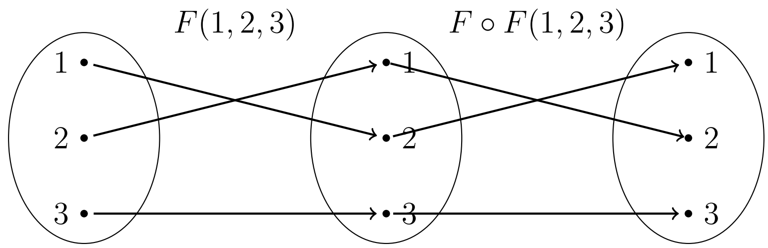


Figure 4.3: All 12 binary strings of length 5 with at least one 1 and no consecutive 1s maps to all twelve ladders of order 6 with a height of one. The recurrence relation being $L(6) = 2L(4) + (L(5) - L(4)) + 1 = L(4) + L(5) + 1$

4.1.2.3 Closed form Formula for Ladders of Order N with a Height of One

Before providing the closed form formula for the number of ladders with a height of one, it is important to connect ladders with a height of one to other mathematical phenomena because ladders with a height follow the same pattern as these other mathematical phenomena. These phenomena include the number of involutions in the Symmetric Group S_N, the Nth Fibonacci meander, and the number of allowable transitions rules for passing from one change to the next in the English art of bell ringing - insert ref. Each of these other mathematical phenomena will be expained along with their connections to ladders of order N with a height of one.

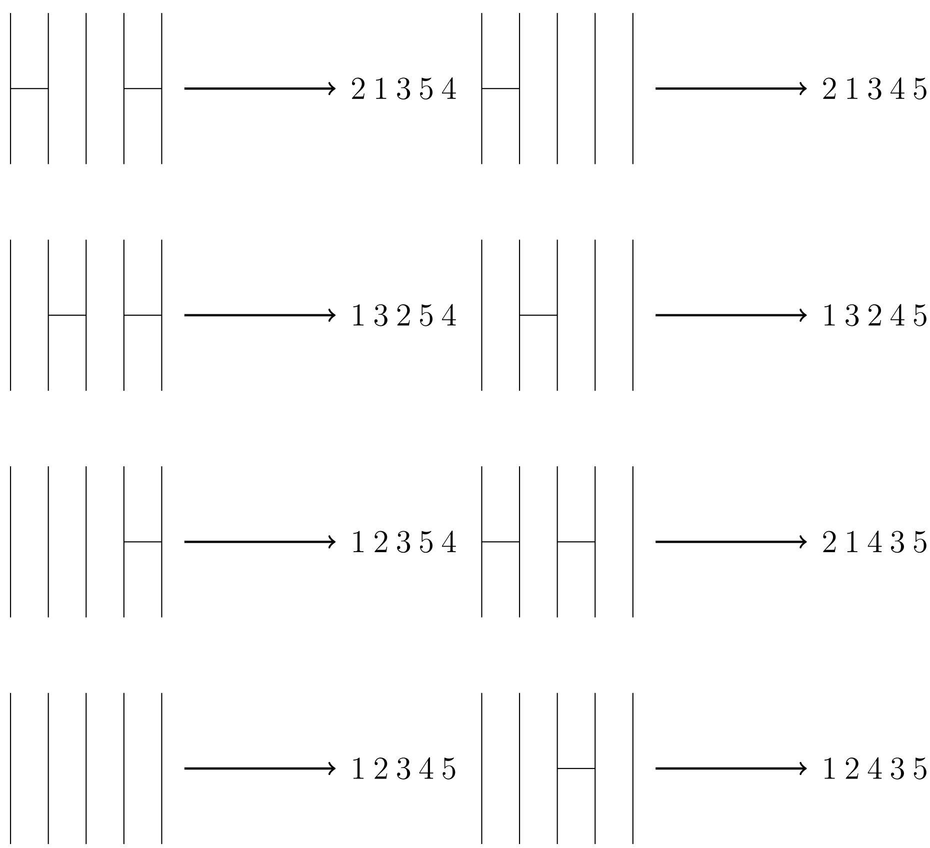
Let S_N be a the symmetric group of degree N such that each of its elements are one of the $N!$ permutations of order N. Let the group operation of S be the composition of two permutations (not necessarily unqique) of order $N-1$. Let an *involution* be defined as a composition of a permutation with itself such that the result of the composition is the identity permutation. For example, $X = \{1, 2, 3\}$. Let $S_X = S_N = \{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)\}$. The involutions of $S_N = \{(2, 1, 3), (1, 3, 2), (1, 2, 3)\}$. The reason these are the involutions is because when we define a permutation as a bijective function on the identity permutation, we can see the the composition of a permutation from the involution set with itself returns the identity permutation. Let $(1, 2, 3) = F$,$(2, 1, 3) = G$ and $(1, 3, 2) = H$ then we have $F \circ F = (1, 2, 3)$, $G \circ G = (1, 2, 3)$ and $H \circ H = (1, 2, 3)$. To see an example of the mapping of the composition of $(2, 1, 3)$ with itself see figure – fig.



Theorem 4.1.4 *There is a surjective function between ladders of order N with a height of one and the involution set of S_N. Note that if one were includes the ladder of order N with a height of zero, then there is a bijective function between ladders of order N with a height of zero or one and the involution set of S_N.*

Proof. The involution set of S_N consists of all permutations of order N such that when composed with themselves, the result of the composition is the identity permutation. If a permutation is an involution it either has no inversions or for each pair of inversions, the inversion pairs are pairwise disjoint. That is to say, no element in the involution forms more than 1 inversion. When inversion pairs are pairwise disjoint, each element in the pair is rotated by one position from its position in the identity permutation. When an involution is composed with the identity permutation, each element is rotated by one or zero positions. If an element from the identity permutation is rotated two times over over a span of two positions, the element returns to its original position in the identity permutation. Thus, composing an involution with itself either rotates an element zero times or it rotates an element twice over a span of two positions, thus placing the element in its original position in the identity permutation.

A ladder of order N with a height of one consists only of bars such that each bar swaps an element in π to its correct position in the identity permutation. Suppose an element, X, needed to be swappped more than once in L_π to reach its position in the identity permutation. This would mean the route of $X > 1$. If that is the case, then the bars of route X require their own rows, seeing as each bar of an element's route cannot be on the same row as any of the other bars of its route. But if that is the case then we have a contradiction seeing as we supposed ladders of order N with a height of one. Then it must be the case that for all ladders of order N with a height of one, each bar in any of these given ladders swaps an elmenet in π to its correct position in the idetity permutation. That is to say, every ladder of order N with a height of 1 sorts a π such that each element in π forms at most one inversion. To see a bijective mapping between ladders of order 5 with a height of zero or one and the involution set of S_5 please refer to figure –fig. $\square$


2 1 3 5 4
2 1 3 4 5
1 3 2 5 4
1 3 2 4 5
1 2 3 5 4
2 1 4 3 5
1 2 3 4 5
1 2 4 3 5

Chapter 5

Evaluation

Chapter 6

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.