Wilfredo Bardales Roncalla

On the Lower Bound for the Maximum Consecutive Sub-Sums Problem

Dissertação de Mestrado

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

Advisor: Prof. Eduardo Sany Laber

Rio de Janeiro
August 2013

Pontifícia Universidade Católica
do Rio de Janeiro

Wilfredo Bardales Roncalla

On the Lower Bound for the Maximum Consecutive Sub-Sums Problem

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

Rio de Janeiro, August 29th, 2013

Wilfredo Bardales Roncalla

Wilfredo Bardales Roncalla graduated from the Universidad Católica San Pablo (Arequipa, Peru) in Informatics engineering; has worked as application's engineer at a Peruvian bank and as teaching assistant for the same university.

Bibliographic data

Bardales Roncalla, Wilfredo

On the Lower Bound for the Maximum Consecutive Sub-Sums Problem / Wilfredo Bardales Roncalla ; Advisor: Eduardo Sany Laber. — 2013.
60 f. : il. ; 30 cm

Dissertação (mestrado)-Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, Rio de Janeiro, 2013.
Inclui referências bibliográficas.

1. Informática – Teses. 2. Algoritmos. 3. Casamento Aproximado de Padrões. 4. Casamento de Padrões Misturados. 5. Vetores de Parikh. 6. Subsequência de Soma Máxima. 7. Somas Máximas. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

To God for you have always been my help and comfort in my moments of solitude, as a tender father.

To my advisor Professor Eduardo Sany Laber for the support, patience, incentive and dedication for the realization of this work.

To my friends, those who motivated me to come to Rio, and those who helped me when I needed the most, especially: Percy, José, Marcelo, Chrystiano, Mairon and Manuel.

To my colleagues of the PUC–Rio, and the people of the Departamento de Informática.

To the CNPq and the PUC–Rio, for the financial support, without which this work would not have been realized.

To my parents, brothers, sisters and family in law for their love, motivation and support.

Finally to my lovely wife Débora María, you are my love and my treasure. Thank you for your patience, even in the most difficult times; without you, none of this would have been possible nor meaningful. To my brave sons, Bernardo Rafael and Felipe de Jesús, your innocence, love and affection bring me happiness and make all endeavors worthy.

Abstract

Bardales Roncalla, Wilfredo; Laber, Eduardo Sany (Advisor). **On the Lower Bound for the Maximum Consecutive Sub-Sums Problem**. Rio de Janeiro, 2013. 60p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The Maximum Consecutive Subsums Problem (MCSP) arises in scenarios of both theoretical and practical interest, for example, approximate pattern matching, protein identification and analysis of statistical data to cite a few. Given a sequence of n non-negative real numbers, The MCSP asks for the computation of the maximum consecutive sums of lengths 1 through n. Since trivial implementations allow to compute the maximum for a fixed length value, it follows that there exists a naive quadratic procedure to solve the MCSP. Notwithstanding the effort devoted to the problem, no algorithm is known which is significantly better than the naive solution. Therefore, a natural question is whether there exists a superlinear lower bound for the MCSP. In this work we report our research in the direction of proving such a lower bound.

Keywords

Algorithms; Approximate Pattern Matching; Jumbled Pattern Matching; Parikh Vectors; Maximum Subsequence Sum; Maximum Sums.

Resumo

Bardales Roncalla, Wilfredo; Laber, Eduardo Sany. **Sobre o Limite Inferior para o Problema das Sub-Somas Consecutivas Máximas**. Rio de Janeiro, 2013. 60p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O Problema das Sub-somas Consecutivas Máximas (MCSP) surge em cenários de interesse teórico e prático, por exemplo, no casamento aproximado de padrões, identificação de proteínas e análise de dados estatísticos apenas para nomear alguns. Dada uma sequência de n números reais não negativos, O MCSP consiste em calcular as somas consecutivas máximas de tamanho 1 até n. Como existem implementações triviais que permitem encontrar o máximo para um comprimento fixo, existe um procedimento quadrático simples que permite resolver o MCSP. Apesar dos esforços dedicados ao problema, não é conhecido nenhum algoritmo significativamente melhor que a solução simples. Portanto, uma pergunta natural é se existe um limite inferior superlinear para o MCSP. Neste trabalho reportamos nossas pesquisas no sentido de provar tal limite.

Palavras–chave

Algoritmos; Casamento Aproximado de Padrões; Casamento de Padrões Misturados; Vetores de Parikh; Subsequência de Soma Máxima; Somas Máximas.

Contents

Deus, Deus meus es tu, ad te de luce vigilo. Sitivit in te anima mea, te desideravit caro mea. In terra deserta et arida et inaquosa,[...] Quoniam melior est misericordia tua super vitas, labia mea laudabunt te. Sic benedicam te in vita mea[...] quia fuisti adiutor meus, et in velamento alarum tuarum exsultabo[...]

Psalmus 63 (62)

1 Introduction

The term “Approximate Pattern Matching” comprises several matching models (39, 20); one of them is the *Jumbled Pattern Matching* (20, 23), also referred to as Permutation Matching or Abelian Pattern Matching (18). It consists (in its decision version) in telling if any permutation of a pattern p exists in a given text $\mathcal{T}$ (assuming a finite alphabet for both strings).

This jumbled pattern matching is equivalent to the *Parikh vector pattern matching*. Consider a finite ordered alphabet $\Sigma = \{a_1, a_2, \ldots, a_\sigma\},\ a_1 < a_2 < \ldots < a_\sigma$, the Parikh vector of a string $s \in \Sigma^*,\ s = s_1, s_2, \ldots, s_n$ is defined as $p(s) = (p_1, p_2, \ldots, p_\sigma)$, where $p(s)$ is the Parikh vector that defines the multiplicities of the characters in s, i.e. $p_i = |\{j|\ s_j = a_i\}|$, for $i = 1, 2, \ldots, \sigma$. For example, for the alphabet $\Sigma = \{a, b, c\}$ the string $s = abaccbabaaa$ has Parikh vector $p(s) = (6^a, 3^b, 2^c)$. In the Parikh vector pattern matching, given a string $\mathcal{T}$ (the text) and a pattern Parikh vector $q = (q_1, q_2, \ldots, q_\sigma)$, we are interested in finding all occurrences of q in $\mathcal{T}$, that is, all substrings $t \in \mathcal{T}$ such that $p(t) = q$. In its decision version (*membership* queries) we are only interested in knowing whether or not q occurs in $\mathcal{T}$. Using the previous example, the Parikh vector $q = (2^a, 1^b, 1^c)$ appears only once in s ($abac$).

Most typically, applications for the Parikh vector pattern matching, are found in computational biology and regard the identification of compounds of complex molecules whose presence can be characterized by the presence of certain substructures and their occurrence within a relatively short distance, whilst the exact location of such substructures within the molecule is not significant (2, 11, 30, 57).

Indeed, like the classical pattern matching, a Parikh vector matching can be found in $O(n)$ time. However, while the classical pattern matching requires clever ideas like the Knuth-Morris-Pratt algorithm (46) or the Boyer-Moore algorithm (13), Parikh vector matching can be easily solved in $O(n)$ time with a simple sliding window based algorithm (23). The difficulty in Parikh vector matching is for the indexing problem (i.e., preprocess a text to efficiently answer Parikh vector queries). For indexing, the classical pattern matching, the text can be preprocessed in $O(n)$ time to produce a data structure of size

$O(n)$, such as a suffix tree (52, 68, 69), that can answer queries efficiently. In contrast, for Parikh vector matching we do not currently have any $o(n^2)$ size data structure for answering queries in time linear in the pattern's size. This motivates the interest in indexes for membership queries which, given several queries over the same string, allows to at least filter out the "no" instances, before, eventually applying the trivial $O(n)$ procedure only to the "yes" instances.

In binary alphabets, it turns out that there is a simple data structure for membership queries that requires only $O(n)$ space to answers queries in $O(1)$ time. This data structure is exactly what the Maximum Consecutive Subsums Problem (MCSP) computes.

Given a sequence of n non-negative numbers, The MCSP asks for the computation of the maximum consecutive sums of lengths 1 through n. This problem has recently generated much interest (16, 17, 18, 23, 24, 55, 56). In particular, for a binary string s of size n, knowing for each $\ell = 1, 2, \ldots, n$ the minimum and maximum number of 1's found in a substring of s of length ℓ, we can answer membership queries in constant time. More precisely the connection between the MCSP and the Parikh vector membership query problem is given by the following (interval) lemma:

Lemma 1 (Burcsi et al. (17))**.** *Let s be a binary string over the binary alphabet $\Sigma = \{0, 1\}$, and let $\mu^{\min}_\ell$, $\mu^{\max}_\ell$ denote the minimum and maximum number of ones in a substring of s of length ℓ. Then, there exists a substring in s with Parikh vector $p = (x_0, x_1)$ if and only if $\mu^{\min}_{x_0+x_1} \le x_1 \le \mu^{\max}_{x_0+x_1}$*

It follows that, after constructing the tables of $\mu^{\min}$'s and $\mu^{\max}$'s (which is equivalent to solving two instances of the MCSP) we can answer in constant time Parikh vector membership queries, i.e., questions asking: "Is there an occurrence of the Parikh vector (x_0, x_1) in s?".

For example, let s = 001010110001100111 and the tables of $\mu^{\min}$'s and $\mu^{\max}$'s as shown in Table 1.1. Then the Parikh vector $p(000000111111) = (6, 6)$ is present in s while $p(000011111111) = (4, 8)$ is not.

ℓ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\mu^{\min}$	0	0	0	1	2	2	2	3	4	4	4	5	6	6	6	7	8	9
$\mu^{\max}$	1	2	3	3	3	4	5	5	5	5	6	7	7	8	8	9	9	9

Table 1.1: $\mu^{\min}$ and $\mu^{\max}$ tables for s = 001010110001100111

Since trivial implementations allow to compute the maximum for a fixed length value, it follows that there exists a naive quadratic procedure to solve the MCSP. Notwithstanding the effort devoted to the problem, no algorithm is known which is significantly better than the naive solution. Therefore, a natural question is whether there exists a (non-trivial) superlinear lower bound for the MCSP. In this work we report our research in the direction of proving such a lower bound.

1.1 Problem Definition

Given a sequence of n non-negative reals $A[1,\dots,n] = \langle a_1, a_2, \dots, a_n \rangle$, the MAXIMUM CONSECUTIVE SUBSUMS PROBLEM asks to find the consecutive subsequence $A[i^*, j^*] = \langle a_{i^*}, a_{i+1}, \dots, a_{j^*} \rangle$ of size $\ell = j^* - i^* + 1$ whose sum $\sum_{k=i^*}^{j^*} A[k]$ is the maximum, for all $\ell = 1, 2, \dots, n$.

For example, consider the following input sequence:

$$A = \begin{array}{|c|c|c|c|c|c|}\hline 4 & 4 & 0 & 2 & 5 & 2 \\ \hline\end{array}$$

Then, the solution for this MCSP input are the maximum consecutive subsums shown in Table 1.2.

Length	Substring	Sum
1	$A[5]$	5
2	$A[1,2]$	8
3	$A[4\dots6]$	9
4	$A[2\dots5]$	11
5	$A[1\dots5]$	15
6	$A[1\dots6]$	17

Table 1.2: Maximum consecutive subsums for the sequence $A = [4, 4, 0, 2, 5, 2]$

Algorithm 1.1 NAIVE-MCSP($A[1,\dots,n]$)

```
1: for $i \leftarrow 1$ to $n$ do
2:     $sum[i] \leftarrow 0$
3:     $max[i] \leftarrow -\infty$
4: end for
5: for $i \leftarrow 1$ to $n$ do
6:     for $j \leftarrow n$ down to $i$ do
7:         $sum[j] \leftarrow sum[j] + A[j - i + 1]$
8:         if $sum[j] > max[i]$ then
9:             $max[i] \leftarrow sum[j]$
10:        end if
11:    end for
12: end for
13: return $max$
```

It's not difficult to come up with simple $\Theta(n^2)$ algorithms, the Naive-MCSP method is an example of such an algorithm, and despite the interest generated by this problem (16, 17, 18, 23, 24, 55, 56) does not exists a better than $O(n^2/\log^2 n)$ algorithm in the literature. Then, a natural question to ask is whether there exists a non-trivial lower bound for the MCSP. In this thesis we attack that problem.

1.2 Related Works

The classical Maximum Sum Subsequence Problem has been generally used for pedagogical purposes to illustrate design techniques and algorithmical thinking.

Given a sequence of n real numbers $A[1,\ldots,n] = \langle a_1, a_2, \ldots, a_n\rangle$, where the segment $A[i,j] = \langle a_i, a_{i+1}, \ldots, a_j\rangle$ is a consecutive subsequence of A, the problem asks to find the segment $A[i^*, j^*] = \langle a_{i^*}, a_{i^*+1}, \ldots, a_{j^*}\rangle$ whose sum $\sum_{k=i^*}^{j^*} A[k]$ is the maximum among all possible segments. The problem was introduced by Grenader (12) as a special one-dimensional version of its two-dimensional counterpart, the Maximum Sum Subarray Problem. It finds applications in pattern matching (37, 58), biological sequence analysis (1), and data mining (33). The Maximum Sum Subsequence Problem can be solved in linear time using Kadane's algorithm (12), and sometimes it's also called the Maximum Sum Segment.

The Maximum Sum Subarray Problem asks to find the submatrix of a given $m \times n,\ m \leq n$, matrix of real numbers, whose sum is the maximum among all the $O(m^2n^2)$ submatrices. The first sub-cubic algorithm was given by Tamaki and Tokuyama (65), and later Takaoka (64) gave a simplified version of sub-cubic time as well.

A natural generalization is the k Maximum Sum Segments that, given a sequence $A[1,\ldots,n]$ and a positive number k, consist in locate the k segments whose sum are the k largest among all possible sums. The k Maximum Sum Segments was first presented by Bae and Takaoka (5) and, after different solutions emerged (5, 6, 7, 10, 22, 50), was optimally solved by Brodal and Jørgensen (15) in $O(n+k)$

Ruzzo and Tompa (62) considered the All Maximum Sum Disjoint Segments that finds all the disjoint segments of maximum sum, and presented an $O(n)$ algorithm.

The Lengh-Constrained Maximum Sum Segment, given a sequence $A[1,\ldots,n]$ and two integers L, U with $1 \leq L \leq U \leq n$, finds the maximum sum segment among all segments of A of lengths in the interval

$[L, U]$. The Lengh-Constrained Maximum Sum Segment was formulated by Huang (38) and was solved optimally in $O(n)$ (31, 50).

For the Parikh vector pattern matching the best constructions for the binary sequences are as follows: Bursci et al. (16, 17, 18) showed an $O(n^2/\log n)$-time algorithms based on the $O(n^2/\log n)$ algorithm of Bremner et al. (14, 21) for computing a $(\min, +)$-convolution, independently Moosa and Rahman (55) obtained the same result by a different use of $(\min, +)$-convolution, then Mossa and Rahman (56) further improved it to $O(n^2/\log^2 n)$ in the RAM model.

Currently faster algorithms than $O(n^2/\log^2 n)$ exist only when the string compresses well using the run-length encoding (4, 35) or when an approximate index, with a running time of $O(n^{1+\varepsilon})$ and the probability of an incorrect answer depending of the choice of ε, is allowed (24).

1.3 Our Contributions

This thesis describes an attempt to prove a superlinear lower bound for the MCSP in the decision tree model.

First we show that a lower bound on the MCSP can be proved by showing a lower bound on a minimum set of outputs that any algorithm that solves the MCSP must know how to distinguish from. This set must be large enough such that a non-trivial lower bound may be derived using a decision tree model of computation. This technique in principle can be applied to any kind of problem for which a set like that exists.

Then, we show an algebraic way to explore this set. We describe some characteristics about its elements, and we prove that its size is at least exponential.

Finally we give some empirical evidence, using deterministic and probabilistic methods, to show that the size of the aforementioned set is $\Omega(\sqrt{n}!)$ which implies a superexponential size and a corresponding superlinear lower bound. We discuss some ways to improve our results pointing out the main limitations of our current approach.

1.4 Organization

The organization of this work is as follows:

Chapter 2 gives a brief overview about the most common techniques or arguments used to determine lower bounds, exemplifying each of the techniques with known lower bounds.

Chapter 3 shows the approach used to give evidences about the lower bound for the MCSP. We define the concept of configuration and feasibility in order to show approximations for the number of feasible configurations, that allow us to infer a lower bound.

Chapter 4 explains the deterministic and probabilistic methods and shows the main empirical results of this work.

To finalize, Chapter 5 summarizes the results found, conjectures the lower bound for the MCSP and discusses future directions for the research of the problem.

2
Basic Concepts

A very important aspect in the study of algorithms is to know about the algorithmic limitations. This limitations are generally expressed as two "opposite" views: when a computational problem can not be solved efficiently (or at least it's not known if it can) and when it is solved optimally. In other words, intractability and lower bounds.

Intractability dates back to the classical undecidable results obtained by Alan Turing and his famous Halting Problem (67), and has given the major unsolved problem in computer science and one of the most important problems in contemporary mathematics, whether $\mathcal{P} \neq \mathcal{NP}$. For a very nice guide to intractability, $\mathcal{NP}$-completeness theory and discussions about $\mathcal{NP}$-complete problems we refer the reader to the book by Garey and Johnson (34).

On the other hand, given that there are some problems that admit efficient solutions, a natural question arises: *What is the best possible way to solve that particular problem?* Equivalently we could ask: *What is the minimum amount of work that we need to do in order to solve a particular problem?* Regardless of how clever any algorithm can possibly be, a lower bound places a limit on how fast a particular problem can be solved. Thus, a lower bound indicates the minimum amount of resources (space and time primarily) required to solve a computational problem.

In this chapter we describe some techniques and examples about how to prove the existence of lower bounds.

2.1
Lower Bounds

The question about lower bounds has been around for a long time, in fact, even without any computer Charles Babbage stated:

> As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise — by what course of calculation can these results be arrived at by the machine in the shortest time?[1]

[1]BABBAGE, C. 1864 *Passages from the Life of a Philosopher*, ch. 8 p. 137.

In that sense, given that we know how to solve a problem efficiently, a prime goal is to determine if we are solving the problem "optimally". In the analysis of algorithms, the running time is generally measured as a function of the worst-case input data (66) i.e. the number of (machine independent) basic computer steps counted as a function of the size of the input (26). Thus, it's said that an algorithm is efficient if it has a polynomial running time with respect to its input size (41). Such an algorithm will be optimal if a matching lower bound can be proved on the worst-case performance of any algorithm for the same problem. That is, to prove the computational limit that any efficient algorithm will have in order to solve that particular problem. We also say that a lower bound is tight if such an algorithm exists (49). The term *theory of algorithms* is often used to refer to this type of worst-case performance analysis (63), and its prime goal is to classify algorithms according to their performance characteristics. A convenient mathematical tool to describe that classification is the asymptotic notation that was re-introduced, advocated and popularized by Knuth (43); nowadays it's a common standard in algorithm theory textbooks (25, 26, 36, 41, 49, 51, 63, 66). This kind of notation is often used in computer science to express upper bounds (O-notation), lower bounds (Ω-notation) and matching lower and upper bounds (Θ-notation); which are defined as follows (43):

Definition 1. Given a function $f(n)$

$O(f(n))$ denotes the set of all $g(n)$ such that there exists positive constants C and n_0 with $|g(n)| \leq Cf(n)$ for all $n \geq n_0$.

$\Omega(f(n))$ denotes the set of all $g(n)$ such that there exists positive constants C and n_0 with $g(n) \geq Cf(n)$ for all $n \geq n_0$.

$\Theta(f(n))$ denotes the set of all $g(n)$ such that there exists positive constants C, C' and n_0 with $Cf(n) \leq g(n) \leq C'f(n)$ for all $n \geq n_0$

Where $O(f(n))$ can be read as "order at most $f(n)$"; $\Omega(f(n))$ as "order at least $f(n)$" and $\Theta(f(n))$ as "order exactly $f(n)$".

2.2 Lower Bounds Techniques

We now present some examples of lower bounds and the arguments or techniques used to prove them.

2.2.1 Trivial Lower Bounds

The most general method to obtain a lower bound is based on counting the number of items in the problem's input that must be processed and the number of output items that need to be produced (49). For example, any algorithm to perform a $n \times n$ matrix multiplication must perform at least $\Omega(n^2)$ operations because it has to read $2n^2$ input values and output n^2 values, but we don't know if this lower bound is tight. So far the best asymptotic running time, at the time of writing this dissertation, is $O(n^{2.3727})$ (71).

Sometimes these trivial lower bounds are tight, for example to generate all the permutations of n distinct numbers any algorithm must be $\Omega(n!)$ because the output size is $n!$, and it's tight because there exist algorithms that spend a constant time generating each permutation (45).

At other times they are too loose to be useful. For example, for the TRAVELING SALESMAN PROBLEM: given n cities and pairwise distances between them, find a tour that passes through each city once, returns to the starting point, and has minimum total length (48, 66); a lower bound is $\Omega(n^2)$ because its input is $n(n-1)/2$ intercity distances and the output is a sequence of $n+1$ cities for an optimal tour (49). This is too loose because it is not known if a polynomial-time algorithm exists to solve this problem. It was even conjectured by Edmonds in 1967 (29) that there is no polynomial-time algorithm to solve it.

Next we present other techniques or arguments that are often used to show non-trivial lower bounds.

2.2.2 Information Theoretic Lower Bound

This technique seeks to establish a lower bound based on the amount of information that an algorithm must produce in order to solve a particular problem (49). To know the amount of "bits of information" acquired during a particular algorithmic process, it's used a comparison model or decision tree model (25). In the decision tree or comparison tree, the leaves represent the solutions for a particular problem, and the internal nodes represent the non-redundant comparisons needed to reach a solution, such that each comparison yields 1 bit of information (44). Let $S(n)$ be the minimum number of comparisons needed to reach any leaf node in the decision tree; that is, the height of the comparison tree constructed optimally without redundant comparisons. Because all comparisons are binary ($\{0, 1\}$) the comparison tree is a binary tree and the number of leaf nodes in the tree is at most $2^{S(n)}$. Thus, let N be the number of leaf nodes in the comparison tree, the number of bits of information acquired during a root to leaf traversal is $\lceil \lg N \rceil$, which establishes the lower bound of $\Omega(\lg N)$.

For example, the lower bound for any comparison-based sorting algorithm to sort an array A of n items is $\Omega(n \lg n)$ comparisons (44). The reasoning is as follows: in the binary tree used to represent the sequence of comparisons, an internal node $(i:j)$ represents the compare operation between the $A[i]$ and $A[j]$ items. The left subtree corresponds to the case where $A[i] \leq A[j]$, and the right subtree to $A[i] > A[j]$. A leaf node $(i_0\ i_1\ i_2\ \ldots\ i_{n-1})$ represents the order of the sorted items: $A[i_0], A[i_1], A[i_2], \ldots, A[i_{n-1}]$. For example, in Figure 2.1 is represented the comparison tree for $n = 3$.

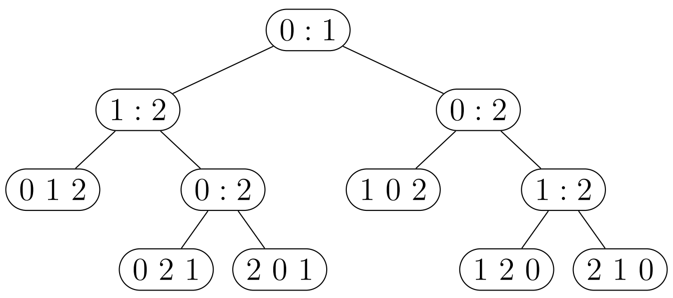


Figure 2.1: Compare tree for $n = 3$, where the internal nodes represent the comparisons and the leaf nodes represent the final ordering

Since all permutations of the n elements are possible and each permutation defines a unique path from the root to a leaf node, it follows that there are exactly $n!$ leaf nodes in a comparison tree that sorts n elements without redundant comparisons. Let $S(n)$ be the minimum number of comparisons that suffices to sort n distinct elements (the height of the comparison tree of n elements), then we can have at most $2^{S(n)}$ leaves. Consequently

$$n! \leq 2^{S(n)}$$

or rewriting this formula:

$$S(n) \geq \lceil \lg n! \rceil$$

Using the Stirling approximation to the factorial function, we have that

$$\lceil \lg n! \rceil = n \lg n - n/\ln 2 + \frac{1}{2} \lg n + O(1)$$

Hence, roughly $n \lg n$ comparisons are needed to sort n elements, establishing the lower bound of $\Omega(n \lg n)$.

Ford and Johnson (32) introduced decision trees as a model to study comparison based sorting algorithms. Later, Rabin (60) generalized the decision trees to the more general class of algebraic computation-trees, and many powerful techniques for linear decision trees were presented (61, 27, 28). Steele and Yao (53) extended the work of Dobkin and Lipton (28) and more generalizations of the decision-tree model were studied comprehensively by Ben-Or (9). For further reference, a wealth of information about algebraic algorithms can be found in the book by Bürgisser, Clausen and Shokrollahi (19).

2.2.3
Adversary Lower Bound

The next non-trivial technique defines the existence of a (pernicious but honest) adversary that, gradually constructs a worst case (consistent) input at the same time that a decision is made by the algorithm; such that it makes the algorithm work as hard as possible, that is, to force the algorithm to make as many decisions as possible (44, 49).

For example, any algorithm for finding both the smallest and the largest of n distinct keys in an unordered list A (Min-Max Selection Problem), requires at least $\lceil \frac{3n}{2} \rceil - 2$ comparisons (it can take one less in the case of n odd). The first solution for this problem using an adversarial argument is due to Pohl (59), where he also gave an optimal algorithm.

The reasoning is as follows (3): assuming that the n keys are distinct, whenever a comparison of two keys is done we say that the larger key is a winner of the comparison, and the smaller is a loser of the comparison. At any point during the execution of the algorithm, any given key is associated to one of the following labels:

Label	*Description*
W	The key has won at least one comparison and never lost.
L	The key has lost at least one comparison and never won.
WL	The key has won and lost at least one comparison.
N	The key has not yet participated in a comparison.

At the beginning of the algorithm all the keys have label N, and as the comparisons are made, the adversary will respond such that the minimum amount of information can be acquired by the algorithm. The answers of the adversary are defined as follows:

The adversary must keep track of the answers given to the algorithm, so that a consistent answer can be given when both keys have a label WL. A way to do this is to assign values to the keys, when they are asked.

It was also proved with an adversarial argument by Pohl (59) that any

Labels of Keys $A[i]$ – $A[j]$	Adversary Answer	New Labels For $A[i]$ – $A[j]$	Bits of Information Given
N–N	$A[i] > A[j]$	W–L	2
W–N or WL–N	$A[i] > A[j]$	W–L or WL–L	1
L–N	$A[i] < A[j]$	L–W	1
W–W	$A[i] > A[j]$	W–WL	1
L–L	$A[i] > A[j]$	WL–L	1
W–L or WL–L or W–WL	$A[i] > A[j]$	No change	0
WL–WL	Consistent with previous answers	No change	0

Table 2.1: Answers of the adversary for the Min-Max Selection Problem

algorithm that finds the largest of n keys in an unordered list require at least $n-1$ bits of information because at least $n-1$ keys must be losers in a comparison. The intuition is that if we don't compare at least $n-1$ keys, there could exist at least one key for which we don't know nothing and that could be the maximum; for the minimum the reasoning is the same. Thus to find the maximum and the minimum (independently) we need at least $2(n-1)$ bits of information.

Assuming that n is even; to count the number of "bits of information" acquired by the algorithm, we have that only 2 bits of information are given when two keys with label N are compared. That can happen at most $n/2$ times, which means that at most $2(n/2) = n$ bits of information are acquired by the algorithm. As noted before, any algorithm needs at least $2(n-1)$ bits of information to find the maximum and the minimum of an unordered list of n keys, that means that the algorithm needs to acquire at least $2n-2-n = n-2$ additional bits of information. For all the other operations the adversary gives at most 1 bit of information. Thus it needs to do at least the following number of comparisons:

$$n - 2 + \frac{n}{2} = \frac{3n}{2} - 2$$

For the case when n is odd, the number the number of comparisons is:

$$n - 2 + \frac{n-1}{2} + 1 = \left\lceil \frac{3n}{2} \right\rceil - 2$$

which establishes the lower bound for the minimum number of comparisons needed.

Asymptotically, the lower bound is $\Omega(n)$, but the use of the adversary argument allows us to establish a true lower bound on the minimum number of comparisons made by any algorithm for this problem. We shall note that by using the more general information theoretic argument (or decision tree model), the lower bound would be of $\Omega(\lg n)$, which is too loose.

2.2.4
Problem Reductions

Another commonly used technique to determine lower bounds is the problem reduction technique. This approach, is commonly used to compare the relative difficulty of different problems. We say that a “problem X is at least as hard as problem Y” if arbitrary instances of problem Y can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to an algorithm that solves problem X (41). To determine the lower bounds we assume that Y has a known tight lower bound. Then, Y is polynomial-time reducible to X ($Y \leq_P X$), or, X is at least as hard as Y.

The consequence of this reduction is that, if $Y \leq_P X$ then any algorithm that solves X can also be used to solve Y, thus, the lower bound is “propagated” from Y to X, because otherwise there would exists a better lower bound than the already known tight lower bound, which would be a contradiction.

For example, the Element Uniqueness Problem (given a set of n real numbers, decide whether two of them are equal) is known to have a lower bound of $\Omega(n \lg n)$ element comparisons using the algebraic decision-tree model (9). We can use this fact to show that the Closest Pair Problem (given a set of n points in the plane identify the pair with minimum separation) is at least as hard as the Element Uniqueness Problem and, therefore, has also a lower bound of $\Omega(n \lg n)$.

Let $x = \{x_1, x_2, \ldots, x_n\}$ be a set of positive real numbers. Corresponding to each number x_i construct a point $p_i = (x_i, 0)$ such that all the constructed points are on the line $y = 0$. This construction takes $O(n)$ time. Let A be an algorithm that solves the closest pair problem for the constructed set and outputs $(x_i, 0)$, $(x_j, 0)$ as the closest pair. Then, there are two equal numbers in the original set x if and only if the distance between x_i and x_j is equal to zero. Thus we have that: Element Uniqueness Problem $\leq_P$ Closest Pair Problem.

3
Toward a Lower Bound for the MCSP

In this chapter we describe the approach taken to try to establish a lower bound for the MAXIMUM CONSECUTIVE SUBSUMS PROBLEM. In order to do so, let's first define the input and the output for the problem. Let $A[1 \dots n]$ be the input sequence for the MCSP formed by n non-negative real numbers. For example:

$$A = \begin{array}{|c|c|c|c|c|c|}\hline 4 & 4 & 0 & 2 & 5 & 2 \\ \hline\end{array}$$

Then, the output for the problem are the maximum consecutive subsums shown in Table 3.1.

Length	Substring	Sum
1	$A[5]$	5
2	$A[1,2]$	8
3	$A[4 \dots 6]$	9
4	$A[2 \dots 5]$	11
5	$A[1 \dots 5]$	15
6	$A[1 \dots 6]$	17

Table 3.1: Maximum consecutive subsums for the sequence $A = [4, 4, 0, 2, 5, 2]$

One way to encode this output is the following: let $R = \{r_1, r_2, \dots, r_n\}$ be the set of substrings of A whose sum is the maximum for the lengths $1, 2, \dots, n$. Thus, by letting $P[i]$, for $i = 1, 2, \dots, n$, be the starting position of the substring r_i, we can uniquely map any solution for the MCSP into an array P of starting positions, which we will call "maximums configuration" or simply "configuration". Sometimes we will use interchangeably $P[i]$ and p_i to denote the starting position of the maximum of length i.

For example, the corresponding maximums configuration for the input sequence $A = [3, 0, 5, 0, 2, 4]$ is:

$$P = \begin{array}{|c|c|c|c|c|c|}\hline 3 & 5 & 1 & 3 & 2 & 1 \\ \hline \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} & \multicolumn{1}{c}{5} & \multicolumn{1}{c}{6}\end{array}$$

In that way, the substring $A[P[k] \ldots P[k]+k-1]$ represents the maximum sum of length k. Thus, we define the configuration P as the output for the MCSP. The entire configuration can be visually represented by a bidimensional grid, where the positions of each maximum segment are shadowed. By convention, the row i, from the top to the bottom, shows the maximum sum of length i. For example, the Figure 3.1 shows the visual representation for the previous configuration P.

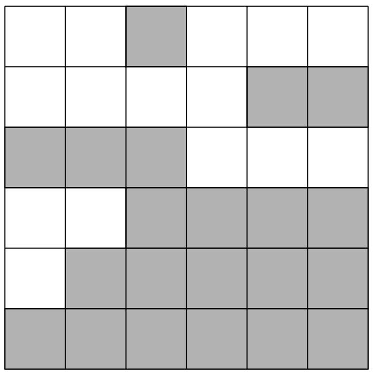


Figure 3.1: Grid representation for the configuration $P = [3, 5, 1, 3, 2, 1]$

We shall note that not all inputs have a unique configuration P as an answer. For example, let $A = [6, 0, 5, 0, 2, 4]$ be the input sequence; in this case the maximum sum of length 2 is 6 and both substrings $A[1, 2]$ and $A[5, 6]$ are valid answers. Similarly, the maximum of length 4 is 11 and both substrings $A[1 \ldots 4]$ and $A[3 \ldots 6]$ are valid answers. Therefore, the answer could be any of the four configurations:

$$P = \begin{cases} [1, 1, 1, 1, 1, 1] \\ [1, 5, 1, 1, 1, 1] \\ [1, 1, 1, 3, 1, 1] \\ [1, 5, 1, 3, 1, 1] \end{cases}$$

Let's discuss about the implications of those multiplicities. Let's suppose that $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ is an arbitrarily chosen set of inputs for the MCSP, and $\mathcal{P}(\mathcal{A}) = \{P \mid P \text{ is a configuration for some } A \in \mathcal{A}\}$. Then, any input $A_i \in \mathcal{A}$ has at least one corresponding solution $P_j \in \mathcal{P}(\mathcal{A})$.

For example, let $\mathcal{A} = \{A_1, A_2, A_3, A_4\}$ and $\mathcal{P}(\mathcal{A}) = \{P_1, P_2, P_3, P_4, P_5, P_6\}$ as shown in Table 3.2, with the following mapping:

$$\begin{aligned}
\mathcal{P}(A_1) &= \{P_1, P_2\} \\
\mathcal{P}(A_2) &= \{P_3, P_4, P_5\} \\
\mathcal{P}(A_3) &= \{P_2, P_3, P_4, P_6\} \\
\mathcal{P}(A_4) &= \{P_5\}
\end{aligned}$$

Input Set $\mathcal{A}$	Solution Set $\mathcal{P}(\mathcal{A})$
$A_1 = [2, 3, 1, 4]$	$P_1 = [4, 3, 2, 1]$
	$P_2 = [4, 1, 2, 1]$
$A_2 = [1, 1, 1, 0]$	$P_3 = [1, 1, 1, 1]$
	$P_4 = [1, 2, 1, 1]$
	$P_5 = [3, 2, 1, 1]$
$A_3 = [1, 0, 0, 1]$	$P_2 = [4, 1, 2, 1]$
	$P_3 = [1, 1, 1, 1]$
	$P_4 = [1, 2, 1, 1]$
	$P_6 = [4, 2, 1, 1]$
$A_4 = [5, 4, 6, 0]$	$P_5 = [3, 2, 1, 1]$

Table 3.2: Arbitrary input set $\mathcal{A}$ and a solution set $\mathcal{P}$

Given that all solutions are equally relevant, an algorithm does not need to answer all the possible solutions (or configurations) for any input sequence A, it is sufficient to answer only one of its solutions. For example, for the inputs A_1, A_2, A_3, A_4 some algorithm could answer P_1, P_3, P_6, P_5, respectively. But that's not the only valid subset of solutions, another algorithm could answer only P_2, P_5, P_2, P_5, respectively.

Definition 2. Given a set system $(U, \mathcal{S})$ consisting of a universe U and a collection $\mathcal{S}$ of subsets of U, a *minimum hitting set* is the smallest subset H of U that intersects (hits) every element $S \in \mathcal{S}$.

In that sense, the minimum hitting set of $\mathcal{P}(A_1) \cup \mathcal{P}(A_2) \cup \ldots \cup \mathcal{P}(A_k)$ determines the minimum number of answers that any algorithm needs to distinguish in order to correctly solve instances $A_1, A_2, \ldots, A_k$. This observation motivates the Proposition 2.

Proposition 2. *If $\mathcal{A}$ is a subset of inputs for the* MCSP *and $\mathcal{H}$ is a minimum hitting set of $\{\bigcup\mathcal{P}(A) \mid A \in \mathcal{A}\}$, then $\log|\mathcal{H}|$ is a lower bound for the running time of the* MCSP *in the decision tree model.*

Proof. Using a information theoretic argument, let $N = |\mathcal{H}|$ be the cardinality of the minimum hitting set $\mathcal{H}$. In order to uniquely distinguish each of the N configurations from the others we need at least $\lceil \lg N \rceil$ bits of information. Thus, the lower bound is $\lceil \lg N \rceil$ because we need at least that number of steps to uniquely identify any solution of the set $\mathcal{H}$ using the decision tree model. □

We know that finding the minimum hitting set is a difficult problem (40) but if we restrict the set of inputs of the MCSP to those inputs with only one solution, then we can trivially find the minimum hitting set, which is the entire solution set; even further, if the solution set derived from this restricted input set is large enough, then it could be used as lower bound for the unrestricted input case because the latter is at least as big as the former.

In our search for the lower bound, we enforce the restriction of unique solutions for the MCSP by considering only inputs for which all their maximums are unique. For example, all the maximums for the input $A = [3, 5, 0, 3, 2, 4]$ are unique, and are shown in Table 3.3. We'll call this kind of input as restricted input.

Length	Substring	Sum
1	$A[2]$	5
2	$A[1, 2]$	8
3	$A[4 \ldots 6]$	9
4	$A[1 \ldots 4]$	11
5	$A[2 \ldots 6]$	14
6	$A[1 \ldots 6]$	17

Table 3.3: Unique maximums for the sequence $A = [3, 5, 0, 3, 2, 4]$

This restriction about the input is the key point to place a lower bound for the general problem, because it will always be true that the number of solutions that any algorithm could possibly answer will be at least as big as the number of solutions for the restricted input case. Thus, a lower bound for the general case of the MCSP would also be shown.

From now on, whenever we refer to the input (unless otherwise specified) we are referring to the restricted input. Even further, we say that a *configuration P is feasible* if there exists an input A such that P is its unique solution, and *infeasible* otherwise.

Unfortunately, there are configurations P for which it does not exist such input sequence A. If every configuration were feasible a clear lower bound of

$\Omega(n \log n)$ would exist for the $n!$ possible configurations in the decision tree model. Let's show why some configurations are infeasible.

For example, consider the following configuration:

$$P = \begin{array}{|c|c|c|c|c|c|} \hline 2 & 3 & 4 & 1 & 2 & 1 \\ \hline \end{array}$$
$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

then, for P to be feasible, it should exist an input sequence A such that

$$\begin{aligned} A[2] &> A[6] && \text{by } P[1] \\ A[3]+A[4] &> A[4]+A[5] && \text{by } P[2] \\ A[4]+A[5]+A[6] &> A[2]+A[3]+A[4] && \text{by } P[3] \end{aligned}$$

which is infeasible because if we add the first two inequalities we show an inequality that contradicts the third one, namely

$$A[2]+A[3]+A[4] > A[4]+A[5]+A[6] \qquad ※$$

which makes the configuration infeasible.

That's because, in order to be unique maximums, each one of the $P[i]$, $i = 1, \dots, n$ positions describes implicitly a set of $n - i$ inequalities of the form

$$\sum_{k=P[i]}^{P[i]+i-1} a_k \geq \varepsilon + \sum_{\substack{k=j \\ j \neq P[i] \\ j=1,\dots,n-i+1}}^{j+i-1} a_k \tag{3-1}$$

where $\varepsilon > 0$ represents a sufficiently small positive number; and the additional n non-negative restrictions

$$\underset{i=1,\dots,n}{a_i} \geq 0 \tag{3-2}$$

Thus, if for a given configuration, the set of implicit inequalities does not hold, then the configuration is not feasible.

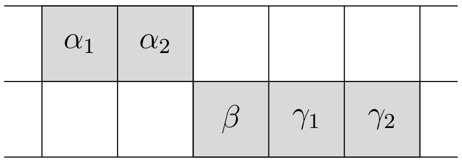
The system of linear inequalities, generated for a configuration P by the Formulas (3-1) and (3-2), describes a linear program. Hence, to test for feasibility is sufficient to solve the linear program using a mathematical programming solver for linear programs. Then, the configuration is feasible if, and only if, a solution can be found for the linear program, and each a_i represents the i-th non-negative number of a input sequence A for which P is a solution.

Thus, in order to determine a lower bound, a natural question to ask is: how many feasible configurations are there?

3.1
Characterizing Feasible Outputs

In order to estimate the number of feasible configurations it is useful to recognize when a given configuration P is not feasible. Next we describe some patterns that can help us to rule out some infeasible configurations.

For example, consider the input sequence $A = [\dots, \alpha_1, \alpha_2, \beta, \gamma_1, \gamma_2, \dots]$ where $\langle \alpha_1, \alpha_2 \rangle$ is the unique maximum of length 2 and $\langle \beta, \gamma_1, \gamma_2 \rangle$ is the unique maximum of length 3:



then it must be true that $\alpha_1 + \alpha_2 > \gamma_1 + \gamma_2$ and also that $\alpha_1 + \alpha_2 + \beta < \gamma_1 + \gamma_2 + \beta$ which is contradictory. Hence, it's impossible to have a feasible solution with the maximum of length 2 adjacent to the maximum of length 3. Generalizing this result we have:

Proposition 3. *There is no feasible configuration such that $p_j = p_i + i$ for some $i,\ j \in \{1, 2, \dots, n\}$, where p_i and p_j are, respectively, the starting positions for the maximums of size i and j.*

Proof. Let p_i be the starting position of the maximum of length i and let p_j the starting position of the maximum of length j adjacent to p_i. Without loss of generality let $p_i < p_j$ meaning that $p_j = p_i + i$ and also let $i < j$.

By definition, the segment of length j starting at p_j is greater than any other segment of length j, specifically greater than the segment starting at p_i:

$$\varepsilon + \sum_{k=p_i}^{p_i+j-1} A[k] \leq \sum_{k=p_j}^{p_j+j-1} A[k]$$

By decomposing the previous inequality

$$\varepsilon + \sum_{k=p_i}^{p_i+i-1} A[k] + \sum_{k=p_i+i}^{p_i+j-1} A[k] \leq \sum_{k=p_j}^{p_j+(j-i)-1} A[k] + \sum_{k=p_j+(j-i)}^{p_j+j-1} A[k]$$

Replacing p_j by $p_i + i$ in the first summand of the right hand side

$$\varepsilon + \sum_{k=p_i}^{p_i+i-1} A[k] + \sum_{k=p_i+i}^{p_i+j-1} A[k] \leq \sum_{k=p_i+i}^{p_i+j-1} A[k] + \sum_{k=p_j+(j-i)}^{p_j+j-1} A[k] \qquad \text{so that}$$

$$\varepsilon + \sum_{k=p_i}^{p_i+i-1} A[k] \leq \sum_{k=p_j+(j-i)}^{p_j+j-1} A[k] \tag{3-3}$$

By definition, for p_i we also have that:

$$\varepsilon + \sum_{k=p_j+(j-i)}^{p_j+j-1} A[k] \leq \sum_{k=p_i}^{p_i+i-1} A[k] \tag{3-4}$$

Adding (3-3) and (3-4) we get

$$2\varepsilon + \sum_{k=p_i}^{p_i+i-1} A[k] + \sum_{k=p_j+(j-i)}^{p_j+j-1} A[k] \leq \sum_{k=p_j+(j-i)}^{p_j+j-1} A[k] + \sum_{k=p_i}^{p_i+i-1} A[k]$$

Wich is infeasible because $2\varepsilon > 0$. $\square$

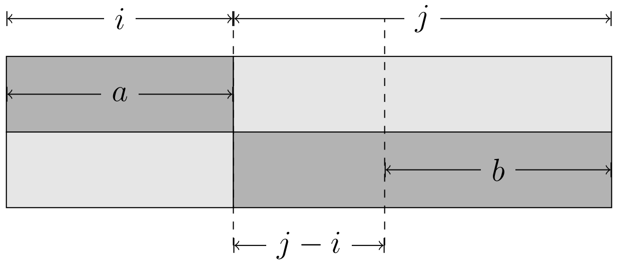


Figure 3.2: Visualization for the proof of Proposition 3, for $p_i < p_j$ and $i < j$.

Basically the proof says that, given two adjacent segments i, j ($p_i < p_j$ and $i < j$) we can define the segments a and b of length i, as shown in Figure 3.2 (the segment a is derived directly from the maximum of length i, and the segment b is derived from the maximum of length j subtracting the overlapping part $j-i$ of the segment of length j starting at p_i). Those adjacent segments would be feasible if, and only if $a > b$ and $b > a$ which is clearly contradictory, making the whole set of inequalities infeasible.

Unfortunately, the non-adjacency does not completely characterize all the configurations that are feasible because there exist configurations with no adjacent maximum that are infeasible. For example, in Figure 3.3 the configuration $P = [2, 4, 2, 1, 2, 1]$ is infeasible because it describes a set of inequalities that doesn't have solution, namely:

$$
\begin{aligned}
A[2] &> A[4] && \text{by } P[1] \\
A[4] + A[5] &> A[1] + A[2] && \text{by } P[2] \\
A[1] + A[2] + A[3] + A[4] &> A[2] + A[3] + A[4] + A[5] && \text{by } P[4]
\end{aligned}
$$

which is infeasible because, if we add the first two inequalities and adding $A[3]$ to both sides, we have an inequality that contradicts the third one

$$A[2] + A[3] + A[4] + A[5] > A[1] + A[2] + A[3] + A[4] \qquad ※$$

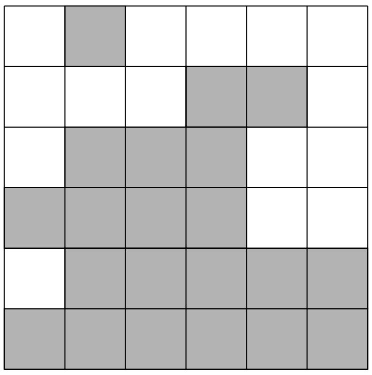


Figure 3.3: Grid representation for the infeasible configuration $P = [2, 4, 2, 1, 2, 1]$ with no adjacent maximums

However, it follows from Proposition 3 that the set of feasible configurations must be a subset of the non-adjacent configurations (configurations with no adjacent maximum), next we show how this concept can help us to understand the size of the set of feasible configurations.

3.2 Approximations for the Number of Feasible Configurations

We've shown that the number of feasible configurations can not be larger than the number of non-adjacent configurations (Proposition 3). If this set is not large enough, such that the log of its size is linear, then none of its subsets could generate a superlinear lower bound and we could only conclude with a trivial lower bound for the number of feasible configurations. However, this is not the case. We show a, large enough, lower bound for the number of non-adjacent configurations. To determine such lower bound we must first define:

Definition 3. A maximum of length i, with starting position p_i, passes through the j-th position if $p_i \leq j \leq p_i + i - 1$.

Using this definition for the case of $\left\lceil \frac{n}{2} \right\rceil$ we can show that:

Proposition 4. *The number of configurations such that every starting position* $p_i,\ i = 1, 2, \ldots, n$ *passes through the position* $\left\lceil \frac{n}{2} \right\rceil$ *is* $\left\lceil \frac{n}{2} \right\rceil! \times \left\lfloor \frac{n}{2} \right\rfloor!$

Proof. Using $j = \left\lceil \frac{n}{2} \right\rceil$ we have that the starting position p_i for the maximums of length i is between:

$$p_i \in \begin{cases} \left[\left\lceil \frac{n}{2} \right\rceil - i + 1,\ \left\lceil \frac{n}{2} \right\rceil\right] & \text{if } 1 \leq i \leq \left\lceil \frac{n}{2} \right\rceil \\ [1,\ n - i + 1] & \text{if } \left\lceil \frac{n}{2} \right\rceil < i \leq n \end{cases}$$

More specifically, the number of possibilities for each maximum i of length $1, 2, \ldots, n$ is $1, 2, \ldots, \left\lceil \frac{n}{2} \right\rceil, \left\lfloor \frac{n}{2} \right\rfloor, \left\lfloor \frac{n}{2} \right\rfloor - 1, \ldots, 1$ respectively. Thus, the total number of possible configurations generated by those maximums is $1 \times 2 \times \ldots \times \left\lceil \frac{n}{2} \right\rceil \times \left\lfloor \frac{n}{2} \right\rfloor \times \left\lfloor \frac{n}{2} \right\rfloor - 1 \times \ldots \times 1 = \left\lceil \frac{n}{2} \right\rceil! \times \left\lfloor \frac{n}{2} \right\rfloor!$. □

Corollary. *The number of non-adjacent configurations is at least* $\Omega\left(\left(\frac{n}{2}!\right)^2\right)$

Proof. Given that all the maximums pass through the $\left\lceil \frac{n}{2} \right\rceil$ position, then all of them share at least one position and no adjacency is generated. □

Therefore, the number of non-adjacent configurations is big enough to contain a subset of super-exponential size, but this does not guarantee that the number of feasible configurations is even exponential. To address this issue we present a constructive proof to create at least $2^{\left\lfloor \frac{n}{2} \right\rfloor}$ feasible configurations.

Proposition 5. *There exists at least* $\Omega\left(2^{\frac{n}{2}}\right)$ *feasible configurations.*

Proof. Let A be the input sequence of size n defined as

$$A[i] = \begin{cases} 0 & \text{if } 1 \leq i < \left\lceil \frac{n}{2} \right\rceil - 1 \\ 2 & \text{if } i = \left\lceil \frac{n}{2} \right\rceil - 1 \\ 4n & \text{if } i = \left\lceil \frac{n}{2} \right\rceil \\ 3 & \text{if } \left\lceil \frac{n}{2} \right\rceil < i \leq n \end{cases}$$

And its unique solution P

$$P[i] = \begin{cases} \left\lceil \frac{n}{2} \right\rceil & \text{if } 1 \leq i \leq \left\lceil \frac{n}{2} \right\rceil \\ n - i + 1 & \text{if } \left\lceil \frac{n}{2} \right\rceil < i \leq n \end{cases}$$

Or more visually:

$$P = \begin{array}{|c|c|c}\hline \left\lceil\frac{n}{2}\right\rceil & \left\lceil\frac{n}{2}\right\rceil & \\ \hline \end{array} \cdots \begin{array}{c|c|c|c}\hline & \left\lceil\frac{n}{2}\right\rceil & \left\lfloor\frac{n}{2}\right\rfloor & \\ \hline \end{array} \cdots \begin{array}{c|c|c|}\hline & 2 & 1 \\ \hline \end{array}$$

$$\begin{array}{ccccc} 1 & 2 & \left\lceil\frac{n}{2}\right\rceil & n-1 & n \end{array}$$

Then, it is possible to create a new configuration with any subset M of maximums of length i, $i = 2, 3, \ldots, 1 + \left\lfloor\frac{n}{2}\right\rfloor$, starting at position $\left\lceil\frac{n}{2}\right\rceil - 1$ by changing the value $A\left[\left\lceil\frac{n}{2}\right\rceil + i - 1\right]$ to 1, for each maximum of length $i \in M$. In that way the sum of length i starting at $\left\lceil\frac{n}{2}\right\rceil$ is dominated by the one starting at $\left\lceil\frac{n}{2}\right\rceil - 1$ because

$$A\left[\left\lceil\frac{n}{2}\right\rceil - 1\right] + A\left[\left\lceil\frac{n}{2}\right\rceil\right] + A\left[\left\lceil\frac{n}{2}\right\rceil + 1\right] + \ldots + A\left[\left\lceil\frac{n}{2}\right\rceil + i - 2\right] >$$
$$A\left[\left\lceil\frac{n}{2}\right\rceil\right] + A\left[\left\lceil\frac{n}{2}\right\rceil + 1\right] + \ldots + A\left[\left\lceil\frac{n}{2}\right\rceil + i - 2\right] + A\left[\left\lceil\frac{n}{2}\right\rceil + i - 1\right]$$

As $\left\lfloor\frac{n}{2}\right\rfloor$ values can take at least 2 different values, then the number of feasible configurations created is $2^{\left\lfloor\frac{n}{2}\right\rfloor}$ and the number of feasible configurations is at least $\Omega\left(2^{\frac{n}{2}}\right)$. $\square$

For example, let $A = [0, 0, 2, 28, 3, 3, 3]$ be the input sequence and its corresponding configuration $P = [4, 4, 4, 4, 3, 2, 1]$, in order to generate the configurations $P = [4, 3, 4, 3, 3, 2, 1]$ $(M = \{2, 4\})$ the input can change accordingly $A = [0, 0, 2, 28, 1, 3, 1]$. In Figure 3.4 is shown the layout for both configurations using the grid representation.

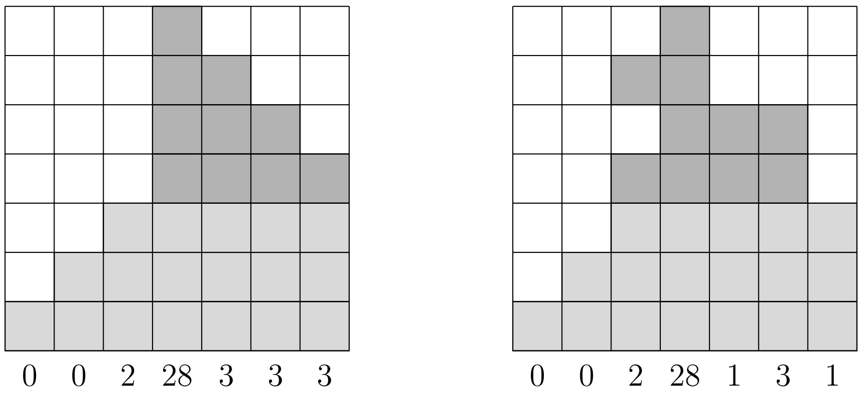


Figure 3.4: Construction of feasible configurations.

In summary we've shown how to determine if a given configuration is feasible or not by representing it as a set of linear inequalities and solving the linear program. We've also shown how to characterize some of those infeasible configurations via an adjacency test without running a optimization solver. Using this characterization we've proved that is possible to have a subset of non-adjacent configurations of super-exponential size, and proved that the number of feasible configurations should be at least exponential. However, we fail to prove the existence of a set of feasible configurations of super-exponential size. Thus, in the next chapter we give empirical evidence that this set is large.

4
Computational Approximations

Having discussed the approach towards the lower bound, next it's presented empirical evidence about the number of feasible configurations.

Given the exponential nature of this problem, the methods are divided in two parts, first it's presented a deterministic way of counting the number of feasible configurations and, as the running time is reasonably big, probabilistic evidences about the number of feasible configurations are presented.

The objective of these experiments is to give an empirical evidence about the number of feasible configurations in order to show a lower bound for the MCSP. To show how large the values are, we'll compare them against the square root of the factorial function ($\sqrt{n!}$ – arbitrarily chosen –) which would also imply a lower bound of $\Omega(n \lg n)$.

All the experiments were performed under the following hardware and software specifications:

– Main Hardware Platform: Intel® Core$^{\text{TM}}$ i7 3960X, 3.30GHz CPU, 32GB RAM, 64-bit.

– Operating System: Windows 7 Professional x64

– Compiler: Microsoft®Visual C# 2010 Compiler version 4.0.30319.1

– Solver: Microsoft Solver Foundation (MSF) 3.1 for the deterministic methods and `lp_solve` 5.5 (interfaced through MSF 3.1) for the randomized methods.

We used the `lp_solve` 5.5 for the randomized methods because as n gets larger (greater than 13) it shows a better performance than the MSF 3.1 solver.

All executions were terminated (at most) after 24 hours of execution time, and it was used the value of $\varepsilon = 1$ for the feasibility tests.

4.1 Deterministic Methods

In order to count the number of feasible configurations deterministically a very large numbers of configurations must be generated, in fact $n!$ configurations are possible and an equal number of linear programs must be solved, each with $\binom{n}{2} + n$ inequalities described by the Formulas (3-1) and (3-2). But, as discussed above, not all of them must be counted nor solved.

Next we present the methods used to deterministically count the number of feasible configurations.

4.1.1 Brute Force (BF)

The Brute-Force(P, i) method is a backtracking algorithm that, given an (initially empty) configuration P of size n, tries all possibilities for the i-th position of P, and continues recursively until $i > n$, then it runs the optimization solver to determine the configuration's feasibility.

For each of the n positions we have $n - i + 1,\ i = 1, 2, \ldots, n$ possibilities for the starting position of the maximum of length i. Hence, a total of $n!$ configurations are possible.

The method Is-Feasible(P) of line 3 receives a configuration P, generates all the implicit inequalities – Formulas (3-1) and (3-2) – described by P, and solves the linear program returning true if it's a feasible configuration or false otherwise. Thus, the execution of Brute-Force$(P, 1)$ counts all feasible configurations of size n.

Algorithm 4.1 Brute-Force(P, i)

```
1: $n \leftarrow |P|$
2: if $i > n$ then
3:     if Is-Feasible$(P)$ then
4:         $count \leftarrow count + 1$
5:     end if
6:     return
7: end if
8: for $j \leftarrow 1$ to $n - i + 1$ do
9:     $P[i] \leftarrow j$
10:    Brute-Force$(P, i + 1)$.
11: end for
```

4.1.2
Brute Force With Level Validation (BFLV)

To reduce the number of linear programs solved by the Brute-Force algorithm it is important to consider that if the infeasibility is detected earlier in the root to leaf path of the backtracking tree, a considerable number of infeasible linear programs could be avoided.

This can be accomplished verifying the linear program every time a new position is fixed in the configuration P. The method Is-Feasible(P, i) of line 8 receives a configuration P, generates the implicit inequalities (up to i) described by P and solves the linear program returning true if it's a feasible configuration or false, otherwise.

Algorithm 4.2 Brute-Force-With-Level-Validation(P, i)

```
1: $n \leftarrow |P|$
2: **if** $i > n$ **then**
3:     $count \leftarrow count + 1$
4:     **return**
5: **end if**
6: **for** $j \leftarrow 1$ **to** $n - i + 1$ **do**
7:     $P[i] \leftarrow j$
8:     **if** Is-Feasible$(P, i)$ **then**
9:         Brute-Force-With-Level-Validation$(P, i + 1)$.
10:    **end if**
11: **end for**
```

4.1.3
Non-Adjacent Maximums Brute Force (NA)

Another way reduce the number of linear programs evaluated is to rule out some infeasible configurations without running the optimization solver; but this process can not be too time demanding because otherwise it would be counterproductive. To detect the non-feasible configurations it's used the fact stated in Proposition 3. To do so, the method Non-Adjacent-Brute-Force maintains two boolean vectors of size n to restrict the starting positions S and the ending positions E. Whenever a new position is fixed, its two adjacent positions are forbidden (the next adjacent starting position and the previous adjacent ending position). To count all the feasible configurations, without generating any adjacent pair of maximums, the method Non-Adjacent-Brute-Force should be called with the S, E vectors initialized with all their elements set as true in the following way: Non-Adjacent-Brute-Force$(P, 1, S, E)$.

Algorithm 4.3 NON-ADJACENT-BRUTE-FORCE(P, i, S, E)

```
1: $n \leftarrow |P|$
2: **if** $i > n$ **then**
3:     **if** IS-FEASIBLE($P$) **then**
4:         $count \leftarrow count + 1$
5:     **end if**
6:     **return**
7: **end if**
8: **for** $j \leftarrow 1$ **to** $n - i + 1$ **do**
9:     **if** $S[j] =$ TRUE **and** $E[j + i - 1] =$ TRUE **then**
10:        $P[i] \leftarrow j$        // $j$ is not adjacent to any other
11:        $(s, e) \leftarrow (S[j + i], E[j - 1])$
12:        $S[j + i] \leftarrow E[j - 1] \leftarrow$ FALSE
13:        NON-ADJACENT-BRUTE-FORCE($P, i + 1, S, E$)
14:        $(S[j + i], E[j - 1]) \leftarrow (s, e)$
15:    **end if**
16: **end for**
```

4.1.4
Non-Adjacent Maximums With Level Validation (NALV)

Finally, we combine the two previous approaches by solving the linear program every time a new position is fixed, reducing the total number of calls (with infeasible configurations) to the optimization solver. To execute the code the vectors S, E should be initialized with all their elements set as TRUE in the following way: NON-ADJACENT-WITH-LEVEL-VALIDATION($P, 1, S, E$).

4.1.5
Results

In Table 4.1 it's summarized the results of the four counting methods, along with the time taken to compute the number of feasible configurations. In Table 4.2 it's possible to compare the number of infeasible configurations computed by each method. As expected, the BRUTE-FORCE method spent most of its time testing infeasible configurations to extract only "a few" valid configurations. Both pruning criteria (BRUTE-FORCE-WITH-LEVEL-VALIDATION and NON-ADJACENT-BRUTE-FORCE) for $n = 11$ are faster than the exhaustive method by an order of magnitude (a factor of more than 20x and 18x respectively) as shown in Table 4.1. For very small values of n the NON-ADJACENT-BRUTE-FORCE method is faster than the BRUTE-FORCE-WITH-LEVEL-VALIDATION method, but as n grows the number of nodes pruned by the latter is more effective than the former, spending less time computing infeasible configurations. It's important to note that the number of feasible

Algorithm 4.4 NON-ADJACENT-WITH-LEVEL-VALIDATION(P, i, S, E)

```
$n \leftarrow |P|$
**if** $i > n$ **then**
    $count \leftarrow count + 1$
    **return**
**end if**
**for** $j \leftarrow 1$ **to** $n - i + 1$ **do**
    **if** $S[j] =$ TRUE **and** $E[j + i - 1] =$ TRUE **then**
        $P[i] \leftarrow j$        // $j$ is not adjacent to any other
        $(s, e) \leftarrow (S[j + i], E[j - 1])$
        $S[j + i] \leftarrow E[j - 1] \leftarrow$ FALSE
        **if** IS-FEASIBLE$(P, i)$ **then**
            NON-ADJACENT-WITH-LEVEL-VALIDATION$(P, i + 1, S, E)$
        **end if**
        $(S[j + i], E[j - 1]) \leftarrow (s, e)$
    **end if**
**end for**
```

n	Feasible Configurations	$\sqrt{n!}$	BF(sec)	BFLV(sec)	NA(sec)	NALV(sec)
1	1	1	<0.1	<0.1	<0.1	<0.1
2	2	1	<0.1	<0.1	<0.1	<0.1
3	4	2	<0.1	<0.1	<0.1	<0.1
4	12	5	<0.1	<0.1	<0.1	<0.1
5	36	11	<0.1	<0.1	<0.1	<0.1
6	148	27	<0.2	<0.2	<0.1	<0.2
7	586	71	1.3	0.7	0.2	0.6
8	2,790	201	13.2	4.3	1.9	3.6
9	13,338	602	159.9	28.8	16.9	24.5
10	71,562	1,905	2,150.8	214.4	169.1	183.4
11	378,024	6,318	31,491.2	1,502.9	1,727.2	1,288.5
12	2,222,536	21,886		11,380.6	19,270.8	9,800.0
13	12,770,406	78,911				77,079.9

Table 4.1: The first two columns shows the number of feasible configurations and the value of $\sqrt{n!}$, the last four columns shows computational time measured in seconds for each method.

n	BF	BFLV	NA	NALV
1	0	0	0	0
2	0	0	0	0
3	2	2	0	0
4	12	8	0	0
5	84	36	4	4
6	572	150	34	22
7	4,454	702	320	186
8	37,530	3,426	2,614	1,094
9	349,542	18,046	22,156	7,124
10	3,557,238	101,696	192,616	45,570
11	39,538,776	590,208	1,746,054	298,056
12		3,638,396	16,742,836	1,952,638
13				13,122,528

Table 4.2: Number of times that the optimization solver is called, in each of the four methods, and determines that a given configuration is infeasible.

configurations is always bigger than $\sqrt{n!}$ and, as n grows, the difference seems to be bigger and bigger.

The Table 4.2 shows the number of times that, for each algorithm, the solver is called and determines that a configuration is infeasible; and (for the Brute-Force-With-Level-Validation and Non-Adjacent-With-Level-Validation methods) gives an asymptotic notion about why the performance of pruning whole branches of the backtracking-tree is superior. The combination of both pruning techniques is stronger (the Non-Adjacent-With-Level-Validation method) and avoids to test almost an order of magnitude (a factor of 8x) infeasible configurations for $n = 12$, and presumably a lot more for $n = 13$.

4.2 Randomized Methods

Given the exponential nature of the exact deterministic counting strategies, next we describe some probabilistic evidence about the number of feasible configurations.

4.2.1 Mean Estimation (ME)

This method tries to estimate the number of feasible configurations by generating independent and identically distributed random configurations; and treating the feasibility test (i.e. to determine if a configuration is feasible or not) as a Bernoulli trial. Then, the number of feasible configurations over the number of configurations sampled can be used to estimate the true mean (as the number of samples tends to the infinite) and to estimate the total number of feasible configurations as a percentage of the complete space of configurations searched.

One important aspect about this kind of simulation is the number of iterations that we must perform in order to approximate the true mean. For this simulation we try to test if the number of simulations is sufficient to estimate the true mean; that is, given a confidence interval and a error criterion, to perform an output analysis for terminating the simulation (8). Let X_i be the indicator random variable:

$$X_i = \begin{cases} 1 & \text{if the configuration } X_i \text{ is feasible} \\ 0 & \text{otherwise} \end{cases}$$

then, the value that we'll try to estimate is the mean, defined as:

$$\theta = \mathrm{E}\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right)$$

where N is the number of all possible distinct configurations. To estimate θ, let R be the number of simulations performed in a given experiment, then the sampled mean is defined as

$$\hat{\theta} = \frac{1}{R}\sum_{i=1}^{R} X_i$$

and the sampled variance is

$$S^2 = \frac{1}{R-1}\sum_{i=1}^{R}\left(X_i - \hat{\theta}\right)^2$$

Thus, it is desired that a sufficiently large sample size, R, be taken to satisfy

$$\Pr\left(|\hat{\theta} - \theta| \leq \varepsilon\right) \geq 1 - \alpha$$

where $100(1-\alpha)\%$ represents the confidence interval and ε the allowed error. Then, to determine the required sample size (and using a normal approximation, i.e. the central limit theorem), the following formula can be used to determine the number of iterations or sample size (8)

$$R \geq \left(\frac{z_{\alpha/2} \times S}{\varepsilon}\right)^2 \tag{4-1}$$

where $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentage point of the standard normal distribution, forming the $100(1-\alpha)\%$ confidence interval for θ by

$$\hat{\theta} - z_{\alpha/2}\frac{S}{\sqrt{R}} \leq \theta \leq \hat{\theta} + z_{\alpha/2}\frac{S}{\sqrt{R}}$$

We compute the value of R at every iteration and stop whenever the number of simulations performed N is greater than R.

Algorithm 4.5 MEAN-ESTIMATION(n, $z_{\alpha/2}$, ε)

```
1: $R \leftarrow \infty$
2: $N \leftarrow 0$
3: $\hat{\theta} \leftarrow 0$
4: $M_2 \leftarrow 0$
5: $S^2 \leftarrow 0$
6: while $R > N$ do
7:     $N \leftarrow N + 1$
8:     $P \leftarrow$ GET-CONFIGURATION($n$)
9:     $X \leftarrow$ IS-FEASIBLE($P$)        // 1 if it's feasible and 0 otherwise
10:    $\delta \leftarrow X - \hat{\theta}$
11:    $\theta \leftarrow \theta + \delta/N$        // computing the sampled mean
12:    $M_2 \leftarrow M_2 + \delta(X - \hat{\theta})$
13:    if $N > 1$ and $\hat{\theta} > 0$ then
14:        $S^2 \leftarrow M_2/(N-1)$        // computing the sampled variance
15:        $R \leftarrow ((z_{\alpha/2} \times \sqrt{S^2})/\varepsilon)^2$
16:    end if
17: end while
```

The MEAN-ESTIMATION method computes the mean and the variance using the online algorithm proposed by Welford (70) and with the given error and confidence interval determines a sufficiently large value for R.

The method IS-FEASIBLE of line 9 receives a configuration P and returns 1 if it's a feasible configuration and 0 otherwise. For this case, the method IS-FEASIBLE runs a non-adjacency validation, before running the optimization solver because, in that way, in $O(n)$ time we can determine if a given configuration have adjacencies which means that, by Proposition 3, it's infeasible. Otherwise, if the configuration is non-adjacent it runs the optimization solver to determine the feasibility of the configuration.

For the method GET-CONFIGURATION of line 8 we used two different spaces of configurations. The first is the complete $n!$ space of configurations and the second is the space of configurations derived by the Proposition 4 and explained below.

Space of configurations n! This space is formed by all the possible configurations. At each iteration, the starting position of the maximum of length

i is determined in line 2 generating a uniformly distributed random number between 1 and $n - i + i$ inclusive.

Algorithm 4.6 GET-CONFIGURATION(n)

```
1: for $i \leftarrow 1$ to $n$ do        // generating a random configuration
2:     $P[i] \leftarrow U(1,\ n - i + 1)$
3: end for
4: return $P$
```

Space of configurations $\left\lceil \frac{\mathbf{n}}{\mathbf{2}} \right\rceil ! \times \left\lfloor \frac{\mathbf{n}}{\mathbf{2}} \right\rfloor !$ This is the space of all the non-adjacent configurations that passes though the $\left\lceil \frac{n}{2} \right\rceil$ position. The numbers are selected randomly from the ranges of the maximums that passes through $\left\lceil \frac{n}{2} \right\rceil$ (Proposition 4).

Algorithm 4.7 GET-CONFIGURATION(n)

```
1: for $i \leftarrow 1$ to $\left\lceil \frac{n}{2} \right\rceil$ do
2:     $P[i] \leftarrow U(\left\lceil \frac{n}{2} \right\rceil - i + 1,\ \left\lceil \frac{n}{2} \right\rceil)$
3: end for
4: for $i \leftarrow \left\lceil \frac{n}{2} \right\rceil + 1$ to $n$ do
5:     $P[i] \leftarrow U(1,\ n - i + 1)$
6: end for
7: return $P$
```

4.2.2 Lower Bounded Mean Estimation (LBME)

This method doesn't try to estimate the number of feasible configurations by estimating the true mean; instead it tries to bound the true mean from below, such that it is at least as big as an specified threshold, in our case $\sqrt{n!}$. In other words, the number of iterations is determined such that, for a small enough error ε, the number of estimated feasible configurations is at least as big $\sqrt{n!}$, within a confidence interval.

Let $\mathcal{C}$ be the size of the configuration search space, and let $\mathcal{C}^L$ be the threshold. From the Formula 4-1 we have that

$$\varepsilon \geq z_{\alpha/2} \frac{S}{\sqrt{R}} \tag{4-2}$$

Then, R must be large enough and S small enough such that

$$(\hat{\theta} - \varepsilon)\mathcal{C} \geq \mathcal{C}^L$$

Algorithm 4.8 LOWER-BOUNDED-MEAN-ESTIMATION(n, $z_{\alpha/2}$, $\mathcal{C}$, $\mathcal{C}^L$)

```
1: $R \leftarrow 0$
2: $\hat{\theta} \leftarrow 0$
3: $M_2 \leftarrow 0$
4: $S^2 \leftarrow 0$
5: $\varepsilon \leftarrow 0$
6: while $(\hat{\theta} - \varepsilon)\mathcal{C} < \mathcal{C}^L$ do
7:     $R \leftarrow R + 1$
8:     $P \leftarrow$ GET-CONFIGURATION($n$)
9:     $X \leftarrow$ IS-FEASIBLE($P$)      // 1 if it's feasible and 0 otherwise
10:    $\delta \leftarrow X - \hat{\theta}$
11:    $\theta \leftarrow \theta + \delta/R$      // computing the sampled mean
12:    $M_2 \leftarrow M_2 + \delta(X - \hat{\theta})$
13:    if $R > 1$ and $\hat{\theta} > 0$ then
14:        $S^2 \leftarrow M_2/(R-1)$      // computing the sampled variance
15:        $\varepsilon \leftarrow z_{\alpha/2}\sqrt{S^2/R}$
16:    end if
17: end while
```

The logic of the LOWER-BOUNDED-MEAN-ESTIMATION method is similar to the MEAN-ESTIMATION method logic. The main difference is that the error is computed according to the Formula 4-2. With this error, the number of iterations and the sampled variance we can determine a lower bound for the number of feasible configurations.

For example, let $\mathcal{C} = n!$, and let $\mathcal{C}^L = \sqrt{n!}$ for $n = 10$, with $\hat{\theta} = 0.01972$ and an error $\varepsilon = 0.01872$ (computed during the the k-th iteration of the LOWER-BOUNDED-MEAN-ESTIMATION algorithm using the Formula 4-2) the number of feasible configurations is bigger than the lower bound $\mathcal{C}^L$ and the execution of the algorithm can be stopped.

The GET-CONFIGURATION method in line 8, uses the same methods specified for the MEAN-ESTIMATION method, selected according to the size of the space of configurations $\mathcal{C}$ used.

4.2.3 Results

The Table 4.3 shows an estimated number of feasible configurations obtained. We can observe that for the "control" known values ($n = 10, \ldots, 13$) shown in Table 4.4 and their corresponding estimates seem to be very accurate. Using this probabilistic estimation for the number of feasible configurations we were able to increment the value of n up to 20. We should note that the explicit sampled mean ($\hat{\theta}$) is not given because it's always greater than (and very close) to the variance, instead is shown the number of feasible configurations found and the number of experiments performed. It's also interesting to note that every time that the allowed error was decremented, the number of experiments incremented accordingly. All the floating point numbers where rounded up to two decimal places, and the maximum execution time was 24 hours.

n	ε	Experiments	Feasibles	S^2	$n!(\hat{\theta} - \varepsilon)$	$n!(\hat{\theta} + \varepsilon)$
10	10^{-3}	74,327	1,465	19.33×10^{-3}	67,944	75,201
11	10^{-4}	3,629,848	34,532	94.36×10^{-4}	376,276	384,259
12	10^{-4}	1,758,527	8,262	46.27×10^{-4}	2,178,663	2,274,463
13	10^{-4}	774,842	1,483	19.63×10^{-4}	11,624,077	12,869,482
14	10^{-5}	34,639,244	31,121	89.97×10^{-5}	77,629,122	79,372,688
15	10^{-5}	14,239,428	5,250	36.96×10^{-5}	470,431,051	496,584,538
16	10^{-5}	5,686,958	1,036	16.42×10^{-5}	3,226,976,502	3,645,432,300
17	10^{-6}	213,625,070	12,572	57.20×10^{-6}	19,992,561,940	20,703,936,797
18	10^{-6}	83,062,414	1,825	21.80×10^{-6}	133,144,821,566	145,949,568,978
19	10^{-7}	2,908,525,970	22,021	75.71×10^{-7}	908,833,543,177	933,162,563,259
20	10^{-7}	978,429,499	2,492	25.47×10^{-7}	5,953,162,185,964	6,439,742,587,599

Table 4.3: Estimated number of feasible configurations with a confidence of 95% ($z_{\alpha/2} = 1.96$) and allowed error of ε for the ME and the $n!$ space of configurations

The Table 4.4 shows the exact and approximate values for the number of configurations for the complete space of configurations. It can be noticed that the estimated number of feasible configurations for $n = 20$ is already greater than $\sqrt{25!}$. This gives an asymptotic idea of how large the number of feasible configurations is with respect to the squared root of the factorial function.

Using a big enough error ε, we were able to improve our estimates up to $n = 25$. The Table 4.5 shows the lower bounded approximation for the minimum number of feasible configurations using the complete search space of $n!$ and $\sqrt{n!}$ as a lower bound to compute the allowable error. Interestingly it seems enough to sample only 9 feasible configurations to obtain a interval of confidence of 99%, the same behavior occurred for an interval of 95% but the number of feasible samples needed were only 6.

Using both approaches (reducing the search space, and allowing a big enough error) we were able to increment the lower bound for the number of feasible configurations up to $n = 28$. The Table 4.6 shows the lower bounded approximation for the minimum number of feasible configurations using the search space of $\left\lceil \frac{n}{2} \right\rceil ! \times \left\lfloor \frac{n}{2} \right\rfloor !$ and $\sqrt{n!}$ as a lower bound to compute the allowable error. In this case the number of sampled feasible configurations varied a little, but it was considerably smaller than the number of feasible configurations sampled for the ME methods.

n	Feasible Configurations	$\sqrt{n!}$	Factor
1	1	1	1.0x
2	2	1	2.0x
3	4	2	2.0x
4	12	5	2.4x
5	36	11	3.3x
6	148	27	5.5x
7	586	71	8.3x
8	2,790	201	13.9x
9	13,338	602	22.2x
10	71,562	1,905	37.6x
11	378,024	6,318	59.8x
12	2,222,536	21,886	101.6x
13	12,770,406	78,911	161.8x
14	≈78,500,905	295,260	265.9x
15	≈483,507,795	1,143,536	422.8x
16	≈3,436,204,401	4,574,144	751.2x
17	≈20,348,249,369	18,859,677	1,078.9x
18	≈139,547,195,272	80,014,834	1,744.0x
19	≈920,998,053,218	348,776,577	2,640.7x
20	≈6,196,452,386,782	1,559,776,269	3,972.7x

Table 4.4: True and estimated number of feasible configurations and how they are compared against $\sqrt{n!}$

n	ε	Experiments	Feasible Conf.	S^2	$n!(\hat{\theta} - \varepsilon)$
10	23.17×10^{-3}	330	9	26.61×10^{-3}	14,897
11	60.37×10^{-4}	1,278	9	69.98×10^{-4}	40,113
12	31.84×10^{-4}	2,427	9	36.96×10^{-4}	251,199
13	18.66×10^{-4}	4,144	9	21.68×10^{-4}	1,904,585
14	11.19×10^{-4}	6,914	9	13.00×10^{-4}	15,943,765
15	18.00×10^{-5}	43,007	9	20.92×10^{-5}	38,333,554
16	17.92×10^{-5}	43,187	9	20.84×10^{-5}	610,779,065
17	44.05×10^{-6}	175,687	9	51.23×10^{-6}	2,551,292,017
18	23.80×10^{-6}	325,210	9	27.67×10^{-6}	24,807,356,419
19	77.92×10^{-7}	993,333	9	90.60×10^{-7}	154,305,371,850
20	17.00×10^{-7}	4,552,503	9	19.77×10^{-7}	673,359,924,340
21	11.59×10^{-7}	6,679,699	9	13.47×10^{-7}	9,637,383,950,966
22	21.76×10^{-8}	35,566,189	9	25.30×10^{-8}	39,819,894,547,344
23	54.09×10^{-9}	143,100,807	9	62.89×10^{-9}	227,626,575,747,434
24	22.60×10^{-9}	342,364,019	9	26.28×10^{-9}	2,283,432,250,195,320
25	36.92×10^{-10}	2,096,189,348	9	42.94×10^{-10}	9,323,644,785,400,530

Table 4.5: Minimum number of possible feasible configurations with a confidence of 99% ($z_{\alpha/2} = 2.58$) and an allowed error of ε for the LBME method using the $n!$ space of configurations and a threshold of $\sqrt{n!}$

n	ε	Experiments	Feasible Conf.	S^2	$\left\lceil \frac{n}{2} \right\rceil ! \times \left\lfloor \frac{n}{2} \right\rfloor ! (\hat{\theta} - \varepsilon)$
10	12.27×10^{-2}	87	23	19.67×10^{-2}	2,040
11	64.65×10^{-3}	196	28	12.31×10^{-2}	6,757
12	77.41×10^{-3}	125	16	11.25×10^{-2}	26,228
13	53.65×10^{-3}	167	13	72.22×10^{-3}	87,791
14	23.45×10^{-3}	419	15	34.60×10^{-3}	313,820
15	15.07×10^{-3}	634	14	21.63×10^{-3}	1,425,107
16	74.74×10^{-4}	1,285	14	10.78×10^{-3}	5,560,936
17	59.92×10^{-4}	1,423	11	76.76×10^{-4}	25,430,004
18	31.19×10^{-4}	2,738	11	40.03×10^{-4}	118,251,795
19	76.20×10^{-5}	12,662	14	11.05×10^{-4}	452,544,058
20	47.41×10^{-5}	18,847	12	63.63×10^{-5}	2,141,639,684
21	19.01×10^{-5}	46,999	12	25.53×10^{-5}	9,442,157,464
22	64.58×10^{-6}	138,395	12	86.70×10^{-6}	35,264,213,994
23	71.61×10^{-6}	108,076	9	83.27×10^{-6}	222,962,946,159
24	15.65×10^{-6}	546,660	11	20.12×10^{-6}	1,025,446,995,013
25	62.70×10^{-7}	1,364,667	11	80.61×10^{-7}	5,339,970,243,861
26	22.49×10^{-7}	3,805,041	11	28.91×10^{-7}	24,896,939,497,429
27	22.71×10^{-7}	3,407,720	9	26.41×10^{-7}	200,723,602,034,240
28	69.63×10^{-8}	11,115,976	9	80.96×10^{-8}	861,470,894,034,408

Table 4.6: Minimum number of possible feasible configurations with a confidence of 99% ($z_{\alpha/2} = 2.58$) and an allowed error of ε for the LBME method using the $\left\lceil \frac{n}{2} \right\rceil ! \times \left\lfloor \frac{n}{2} \right\rfloor !$ space of configurations and a threshold of $\sqrt{n!}$

4.2.4 Random Walk in the Backtracking Tree

So far, the randomized methods have tried to estimate the number of feasible configurations as a percentage of the configuration space size. Another approach is to try to estimate the number of feasible configurations using the backtracking tree (namely the number of leaves of the tree) and the probability of ending up in a particular leaf.

Let's first remember the BRUTE-FORCE-WITH-LEVEL-VALIDATION method. In this method, every time we assign the value $P[i] = k$ for $k = 1, \ldots, n-i+1$ we test if the configurations is still feasible; if so, we recursively test for $P[i+1]$ for $i = 1, \ldots, n$. If the whole configuration is feasible or infeasible at any point, then the method *backtracks* to a previously unexplored state and continues from there. This process describes a tree which is often called backtracking tree.

The Figure 4.1 shows the backtracking tree for the configurations of size $n = 4$. For example, the configuration $P = [1, 2, x, x]$ is infeasible because the maximums of size 1 and size 2 are adjacent; therefore the children of that internal node does not need to be explored.

In the previous case the exploration of the tree is done via a *depth first* approach. For our new randomized approach we propose a more *breadth first* like traversal. That is, for each level i we first discover all values $P[i] = k$ for $k = 1, \ldots, n-i+1$ such that P is still feasible. Let's say that exists b_i feasible values, we call b_i the *feasible branching factor*, then we choose randomly one of those values for $P[i]$ and continue recursively with $P[i+1]$; if the method observes the feasible branching factors $b_1, b_2, \ldots, b_n$, in a root to leaf path, the method outputs

$$X = \prod_{i=1}^{n} b_i$$

This value will be used to estimate the number of leaves as we explain later.

The Figure 4.2 shows the backtracking tree composed only by the feasible configurations. It also shows the value X for each leaf of the tree, for example the value assigned for the dashed path $2-1-2-1$ is $X = 16$, which is the product of their corresponding feasible branching factors $4, 2, 2, 1$.

The method Branching-Product computes the value X for a random root to leaf path in the backtracking tree T of configurations of size n in a non-recursive way.

Algorithm 4.9 Branching-Product(n)

```
1: $b_p \leftarrow 1$
2: for $i \leftarrow 1$ to $n$ do
3:     $k \leftarrow 1$
4:     for $j \leftarrow 1$ to $n-i+1$ do
5:         $P[i] \leftarrow j$
6:         if Is-Infeasible($P, i$) then
7:             continue
8:         end if
9:         $F[k] \leftarrow j$
10:        $k \leftarrow k+1$
11:    end for
12:    $b_p \leftarrow b_p \times (k-1)$
13:    $P[i] \leftarrow F[U(1, k-1))]$
14: end for
15: return $b_p$
```

The Is-Infeasible method in line 6 returns TRUE if the configuration is infeasible and FALSE otherwise, and all the feasible values are stored in the vector F. Then, a random feasible value is selected from vector F using the function $U(1, k-1)$ of line 13 that returns a random number between 1 and $k-1$ inclusive.

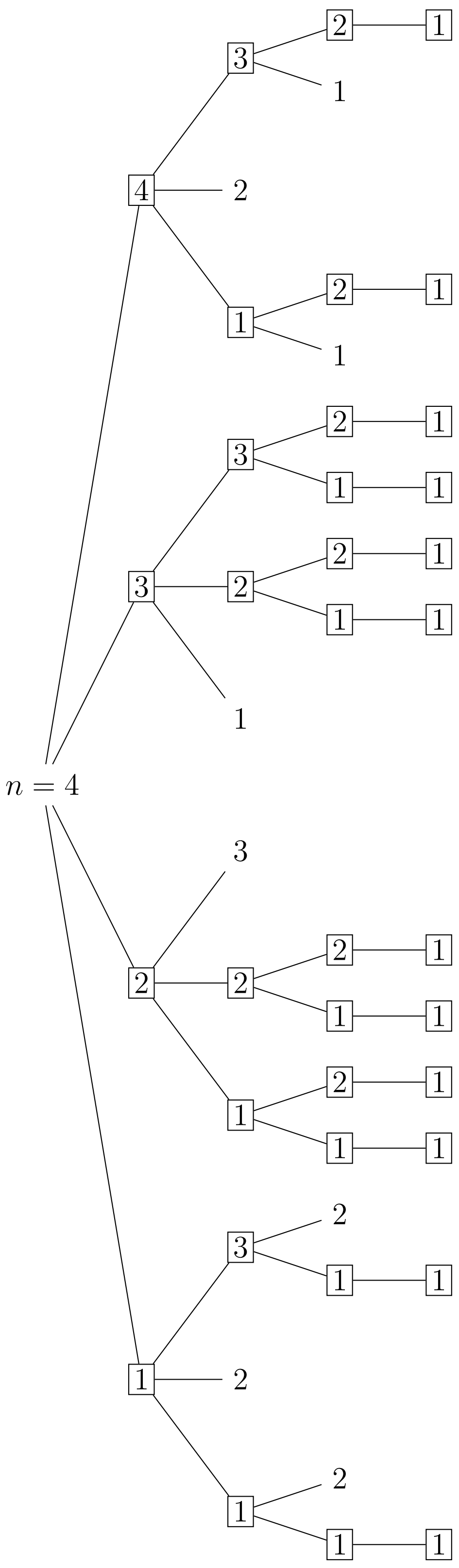


Figure 4.1: Backtracking tree for configurations of size $n = 4$, at each level the feasible positions are shown inside a square.

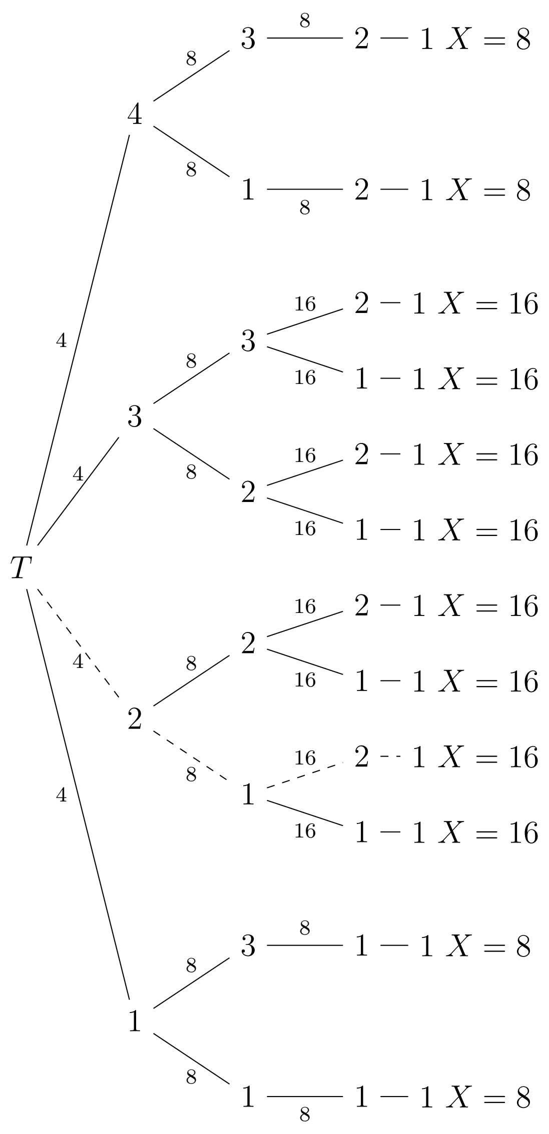


Figure 4.2: Tree induced by the feasible configurations of the backtracking tree $T(4)$, and the value X assigned each one of its leaves. The path $2-1-2-1$ is also show dashed.

Let X be the output of BRANCHING-PRODUCT. Clearly, X is a random variable. We argue that $\mathrm{E}[X]$ is equal to the number of leaves of the tree. In fact, let ℓ be a leaf of the backtracking tree. The probability of a random walk reaching ℓ is $1/B(\ell)$, where $B(\ell)$ is the product of the feasible branching factors in the path from the root of the tree to ℓ. In addition, if ℓ is reached, the method outputs $B(\ell)$. Let L be the set of leaves of the backtracking tree. Thus we have that

$$\mathrm{E}[X] = \sum_{\ell \in L} \frac{1}{B(\ell)} \times B(\ell) = |L|$$

For example, the Figure 4.2 shows the probability distribution, for the backtracking tree $T(4)$, $\{8 \mapsto \frac{1}{2}, 16 \mapsto \frac{1}{2}\}$ and the expected value $\mathrm{E}[X] = 8 \cdot \frac{1}{2} + 16 \cdot \frac{1}{2} = 12$ which is the number of leaves.

After coming up with this approach, we found that it has already been proposed to study heuristics for backtracking algorithms (47, 42).

We must note that we can always reach a leaf node, and therefore the sum of probabilities be 1, as a consequence of the following Proposition

Proposition 6. *Given the first m maximums of a configuration of size n, $m < n$, it is always possible to find a maximum of length $m + 1$.*

Proof. Inductively, given the first m feasible positions of the backtracking tree $T(n)$ it is always possible to create an input, such that, the first m positions are the same, and a feasible position $m + 1$ exists. Consider the input A that the optimization solver answer for the first m maximums, then either the configurations have a unique maximum of length $m + 1$ or not. If the input have a unique maximum of length $m + 1$ the Proposition is proved, if not, it must have more than one maximum. Let δ be the smallest difference between the maximum and the second maximum for lengths $j = 1, \dots, m$. If we add the value $\frac{\delta}{2}$ to the first position of the leftmost maximum, then the new maximum is unique because it is greater than any other maximum of length $m+1$ by at least $\frac{\delta}{2}$; and all the previous maximums remain the same because the value added is less than the smallest difference even if we add several times, at different levels, to the same position $\frac{\delta}{2} + \frac{\delta}{2^2} + \frac{\delta}{2^4} + \dots + \frac{\delta}{2^k} < \delta$. Then, the Proposition is proved. □

We do not use directly the value of X to estimate the number of feasible configurations; instead, we assume that the $\mathrm{E}[X] \leq \sqrt{n!}$ and use the value of X to reject this hypothesis with some level of confidence.

Suppose that $\mathrm{E}[X] \leq \sqrt{n!}$, then by Markov's inequality (54) we have that:

$$\begin{aligned} \Pr[X \geq c\,\mathrm{E}[X]] &\leq \frac{1}{c} \\ \Pr[X \geq c\sqrt{n!}] &\leq \Pr[X \geq c\,\mathrm{E}[X]] \\ \Pr[X \geq c\sqrt{n!}] &\leq \frac{1}{c} \qquad (4\text{-}3) \end{aligned}$$

which imply that

$$\Pr[X < c\sqrt{n!}] \geq 1 - \frac{1}{c} \qquad (4\text{-}4)$$

In other words, under the hypothesis that $\mathrm{E}[X] \leq \sqrt{n!}$, if we sample X and find a value larger than $c\sqrt{n!}$, either we were lucky or the hypothesis is false and we could reject it with confidence of $1 - \frac{1}{c}$. Then, if we reject the hypothesis, the only other possibility is that $\mathrm{E}[X]$ is greater or equal than $\sqrt{n!}$

For example, imagine that, for $n = 10$, we sample a value $X = 46{,}080$; as $46{,}080/\sqrt{10!} \approx 24.19$, letting $c = 24$ by (4-4), with a probability of at least $1 - \frac{1}{24} \approx 95\%$, we must have observed a value less than $24\sqrt{n!}$, however we observed a value greater than that. Therefore, we reject the hypothesis that $\mathrm{E}[X] \leq \sqrt{10!}$ with 95% level of confidence. We know that in this case $\mathrm{E}[X] = 71{,}562$ which validates our rejection.

We extend this approach by taking the maximum of k samples. Let $X_1, X_2, \ldots, X_k$ be the values for k samples of the random variable X. Under the assumption that $\mathrm{E}[X] \leq \sqrt{n!}$ we have that

$$\begin{aligned}
\Pr[\max\{X_1, X_2, \ldots, X_k\} \geq c\sqrt{n!}] &= \Pr[X_1 \geq c\sqrt{n!} \vee \ldots \vee X_k \geq c\sqrt{n!}] \\
&= 1 - \Pr[X_1 < c\sqrt{n!} \wedge \ldots \wedge X_k < c\sqrt{n!}] \\
&= 1 - \prod_{i=1}^{k} \Pr[X_i < c\sqrt{n!}] \\
&\leq 1 - \left(1 - \frac{1}{c}\right)^k \\
\Pr[\max\{X_1, X_2, \ldots, X_k\} \geq c\sqrt{n!}] &\leq 1 - \left(1 - \frac{1}{c}\right)^k \qquad (4\text{-}5)
\end{aligned}$$

Wich imply

$$\Pr[\max\{X_1, X_2, \ldots, X_k\} < c\sqrt{n!}] \geq \left(1 - \frac{1}{c}\right)^k \qquad (4\text{-}6)$$

In other words, assuming that $\mathrm{E}[X] \leq \sqrt{n!}$, and using the $\max\{X_1, X_2, \ldots, X_k\}$, the probability of all the samples being less than $c\sqrt{n!}$, for a large enough c, is very high, and if in those k values we find a value greater or equal than that, then we could reject the hypothesis, which would imply that $\mathrm{E}[X] \geq \sqrt{n!}$.

For example, for $n = 10$ and $k = 10$, imagine that $\max\{X_1, X_2, \ldots, X_{10}\} = 378000$ which is $\approx 198.431\sqrt{10!}$ that means that the probability

$$\Pr[\max\{X_1, X_2, \ldots, X_{10}\} < c\sqrt{10!}] \geq \left(1 - \frac{1}{198}\right)^{10} \approx 95.07\%$$

This implies that we can reject the hypothesis with a confidence of 95.07% because we've found a value greater than $198\sqrt{10!}$. But, if the number of samples were 100 and the maximum remains the same, the confidence level would drop to 60.27%, this clearly shows the trade off between the number values sampled and how this affect the confidence level.

It is important to note that even if we decide not to reject the hypothesis, for example if the confidence level were less than 50%, it does not validate the hypothesis that $\mathrm{E}[X] \leq \sqrt{n!}$.

4.2.5 Results

For this experiment the allowed running time was 16 hours, and the number of experiments or samples was arbitrarily chosen to be $k = 1{,}000$. Each experiment represents a call to the BRANCHING-PRODUCT method. Let c be the floor of the ratio between the maximum value found in the 1,000 samples and $\sqrt(n!)$, Table 4.7 shows the probability of $\Pr[\max\{X_1, X_2, \ldots, X_k\} \leq c\sqrt{n!}]$ assuming that $\mathrm{E}[X] \leq \sqrt{n!}$ for configurations up to size $n = 60$. This probability also expresses our confidence to reject the hypothesis because in fact we've found a value greater than $c\sqrt{n!}$. It can be observed that the probability of finding a big enough value (larger than $\sqrt{n!}$) decreases as n increases, as shown in the third column of the Table.

n	$\left\lfloor \mathrm{MaxFound}/\sqrt{n!} \right\rfloor$	Nº values $X_k \geq \sqrt{n!}$	$\Pr[\max\{X_1, \ldots, X_k\} \leq c\sqrt{n!}]$
10	555	1,000	16.473337%
11	1,474	1,000	50.729819%
12	1,678	1,000	55.094229%
13	9,686	1,000	90.190406%
14	16,101	999	93.977962%
15	14,143	999	93.173312%
16	19,603	999	95.026546%
17	60,931	996	98.372180%
18	137,280	998	99.274205%
19	139,566	996	99.286051%
20	329,058	998	99.696563%
30	75,202,827	965	99.998670%
40	27,078,377	795	99.996307%
50	3,324,405	435	99.969923%
60	1,012,995	145	99.901331%

Table 4.7: Probability of $\Pr[\max\{X_1, \ldots, X_k\} \leq c\sqrt{n!}]$ for $k = 1{,}000$

With this new approach we managed to state that the number of feasible configurations is larger than $\sqrt{n!}$, for $n = 60$ with 99.9% level of confidence.

To be able to report results for larger values of n we need a larger number of samples, each with a quadratic number of calls to the optimization solver, which means that the computational effort would also be increased considerably. Some ideas extend this approach for larger values of n and future research directions are given in the following chapter.

5
Conclusions

This thesis have described an attempt to prove a superlinear lower bound for the MCSP in the decision tree model using an algebraic view of the problem.

We've given a new technique specifying a restricted input case (inputs with unique solutions or feasible configurations) and we've shown that a lower bound over the number of feasible configurations is sufficient to derive a lower bound for the MCSP.

We've presented empirical evidences that the number of feasible configurations is larger than $\sqrt{n!}$. This observation lead us to conjecture that the lower bound for the MCSP is $\Omega(n \lg n)$.

We believe we can show experimental evidences that the number of feasible configurations is larger than $\sqrt{n!}$ for larger values of n. Currently the main limitation is the computational time, that's why future research direction consider to change the current solvers and the interfacing model to a more robust linear optimization tool, allowing parallelization for the feasibility tests at each level of the backtracking tree.

Another promising direction is to implement a probabilistically exploration by approximating the number of feasible configurations at each level of the backtracking tree instead of testing all possible values. Choosing a more suitable factorial function to compare the number of feasible configurations with, also seems a reasonable idea.

Even further, the possibility of a non-empirical lower bound could be achieved if a characterization of the minimum and the maximum branching factors were possible, therefore, future directions also include to explore analytically, deterministically and probabilistically these root to leaf paths of the backtracking tree.

Bibliography

[1] ALLISON, L. **Bioinformatics**. Longest biased interval and longest non-negative sum interval, journal, v.19, n.10, p. 1294–1295, 2003.

[2] AMIR, A.; APOSTOLICO, A.; LANDAU, G. M. ; SATTA, G. **J. of Discrete Algorithms**. Efficient text fingerprinting via parikh mapping, journal, v.1, n.5-6, p. 409–421, Oct. 2003.

[3] BAASE, S. **Computer Algorithms: Introduction to Design and Analysis**. Addison-Wesley series in computer science. Addison-Wesley Pub. Co., 1988.

[4] BADKOBEH, G.; FICI, G.; KROON, S. ; LIPTÁK, Z. **CoRR**. Binary jumbled string matching: Faster indexing in less space, journal, v.abs/1206.2523, 2012.

[5] BAE, S. E.; TAKAOKA, T. **Algorithms for the problem of k maximum sums and a vlsi algorithm for the k maximum subarrays problem.** In: ISPAN, p. 247–253. IEEE Computer Society, 2004.

[6] BAE, S. E.; TAKAOKA, T. **Improved algorithms for the k-maximum subarray problem for small k**. In: PROCEEDINGS OF THE 11TH ANNUAL INTERNATIONAL CONFERENCE ON COMPUTING AND COMBINATORICS, COCOON'05, p. 621–631, Berlin, Heidelberg, 2005. Springer-Verlag.

[7] BAE, S. E.; TAKAOKA, T. **Comput. J.** Improved algorithms for the k-maximum subarray problem, journal, v.49, n.3, p. 358–374, 2006.

[8] BANKS, J.; CARSON, J.; NELSON, B. L. ; NICOL, D. **Discrete-Event System Simulation**. 4. ed., Prentice Hall, Dec. 2004.

[9] BEN-OR, M. **Lower bounds for algebraic computation trees**. In: PROCEEDINGS OF THE FIFTEENTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, p. 80–86. ACM, 1983.

[10] BENGTSSON, F.; CHEN, J. **Efficient algorithms for k maximum sums**. In: Fleischer, R.; Trippen, G., editors, ALGORITHMS AND

COMPUTATION, 15TH INTERNATIONAL SYMPOSIUM, ISAAC 2004, HONGKONG, CHINA, DECEMBER 20-22, 2004, PROCEEDINGS, volume 3341 of **Lecture Notes in Computer Science**, p. 137–148. Springer, 2004.

[11] BENSON, G. **Composition alignment**. In: Benson, G.; Page, R. D., editors, ALGORITHMS IN BIOINFORMATICS, volume 2812 of **Lecture Notes in Computer Science**, p. 447–461. Springer Berlin Heidelberg, 2003.

[12] BENTLEY, J. L. **Programming pearls**. Addison-Wesley, 1986, I-VIII, 1-195p.

[13] BOYER, R. S.; MOORE, J. S. **Commun. ACM**. A fast string searching algorithm, journal, v.20, n.10, p. 762–772, Oct. 1977.

[14] BREMNER, D.; CHAN, T. M.; DEMAINE, E. D.; ERICKSON, J.; HURTADO, F.; IACONO, J.; LANGERMAN, S. ; TASLAKIAN, P. **Necklaces, convolutions, and x + y**. In: PROCEEDINGS OF THE 14TH CONFERENCE ON ANNUAL EUROPEAN SYMPOSIUM - VOLUME 14, ESA'06, p. 160–171, London, UK, UK, 2006. Springer-Verlag.

[15] BRODAL, G. S.; JØRGENSEN, A. G. **A linear time algorithm for the k maximal sums problem**. In: PROCEEDINGS OF THE 32ND INTERNATIONAL CONFERENCE ON MATHEMATICAL FOUNDATIONS OF COMPUTER SCIENCE, MFCS'07, p. 442–453, Berlin, Heidelberg, 2007. Springer-Verlag.

[16] BURCSI, P.; CICALESE, F.; FICI, G. ; LIPTÁK, Z. **On table arrangements, scrabble freaks, and jumbled pattern matching**. In: Boldi, P.; Gargano, L., editors, FUN WITH ALGORITHMS, volume 6099 of **Lecture Notes in Computer Science**, p. 89–101. Springer Berlin Heidelberg, 2010.

[17] BURCSI, P.; CICALESE, F.; FICI, G. ; LIPTÁK, Z. **International Journal of Foundations of Computer Science**. Algorithms for jumbled pattern matching in strings, journal, v.23, n.02, p. 357–374, 2012.

[18] BURCSI, P.; CICALESE, F.; FICI, G. ; LIPTÁK, Z. **Theory of Computing Systems**. On approximate jumbled pattern matching in strings, journal, v.50, n.1, p. 35–51, 2012.

[19] BÜRGISSER, P.; CLAUSEN, M. ; SHOKROLLAHI, M. A. **Algebraic complexity theory**, volume 315. Springer, 1997.

[20] BUTMAN, A.; ERES, R. ; LANDAU, G. M. **Information Processing Letters**. Scaled and permuted string matching, journal, v.92, n.6, p. 293 – 297, 2004.

[21] CHEN, Y.; LU, H.-I. ; TANG, C. **Disjoint segments with maximum density**. volume 3515 of **Lecture Notes in Computer Science**, p. 845–850. Springer Berlin Heidelberg, 2005.

[22] CHENG, C.-H.; CHEN, K.-Y.; TIEN, W.-C. ; CHAO, K.-M. **Improved algorithms for the k maximum-sums problems**. In: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON ALGORITHMS AND COMPUTATION, ISAAC'05, p. 799–808, Berlin, Heidelberg, 2005. Springer-Verlag.

[23] CICALESE, F.; FICI, G. ; LIPTÁK, Z. **Searching for jumbled patterns in strings**. p. 105–117, 2009. cited By (since 1996)13.

[24] CICALESE, F.; LABER, E.; WEIMANN, O. ; YUSTER, R. **Near linear time construction of an approximate index for all maximum consecutive sub-sums of a sequence**. In: Kärkkäinen, J.; Stoye, J., editors, COMBINATORIAL PATTERN MATCHING, volume 7354 of **Lecture Notes in Computer Science**, p. 149–158. Springer Berlin Heidelberg, 2012.

[25] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. ; STEIN, C. **Introduction to Algorithms, Third Edition**. 3rd. ed., The MIT Press, 2009.

[26] DASGUPTA, S.; PAPADIMITRIOU, C. H. ; VAZIRANI, U. V. **Algorithms**. McGraw-Hill Education (India) Pvt Limited, 2006.

[27] DOBKIN, D. **Journal of Computer and System Sciences**. A nonlinear lower bound on linear search tree programs for solving knapsack problems, journal, v.13, n.1, p. 69 – 73, 1976.

[28] DOBKIN, D. P.; LIPTON, R. J. **Journal of Computer and System Sciences**. On the complexity of computations under varying sets of primitives, journal, v.18, n.1, p. 86–91, 1979.

[29] EDMONDS, J. **Journal of Research of the National Bureau of Standards: Mathematics and Mathematical Physics**. Optimum branchings, journal, v.71B, n.4, p. 233–240, October-December 1967.

[30] ERES, R.; LANDAU, G. M. ; PARIDA, L. **Journal of Computational Biology**. Permutation pattern discovery in biosequences., journal, v.11, n.6, p. 1050–1060, 2004.

[31] FAN, T.-H.; LEE, S.; LU, H.-I.; TSOU, T.-S.; WANG, T.-C. ; YAO, A. **An optimal algorithm for maximum-sum segment and its application in bioinformatics**. In: PROCEEDINGS OF THE 8TH INTERNATIONAL CONFERENCE ON IMPLEMENTATION AND APPLICATION OF AUTOMATA, CIAA'03, p. 251–257, Berlin, Heidelberg, 2003. Springer-Verlag.

[32] FORD, LESTER R., J.; JOHNSON, S. M. **The American Mathematical Monthly**. A tournament problem, journal, v.66, n.5, p. pp. 387–389, 1959.

[33] FUKUDA, T.; MORIMOTO, Y.; MORISHITA, S. ; TOKUYAMA, T. **ACM Trans. Database Syst.** Data mining with optimized two-dimensional association rules, journal, v.26, n.2, p. 179–213, June 2001.

[34] GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability; A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman & Co., 1990.

[35] GIAQUINTA, E.; GRABOWSKI, S. **Inf. Process. Lett.** New algorithms for binary jumbled pattern matching, journal, v.113, n.14-16, p. 538–542, July 2013.

[36] GONNET, G. H.; BAEZA-YATES, R. **Handbook of algorithms and data structures: in Pascal and C (2nd ed.)**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1991.

[37] GRENANDER, U. **Pattern Analysis**. New York : Springer-Verlag, 1978.

[38] HUANG, X. **Bioinformatics**. An algorithm for identifying regions of a dna sequence that satisfy a content requirement, journal, v.10, n.3, p. 219–225, 1994.

[39] JOKINEN, P.; TARHIO, J. ; UKKONEN, E. **Softw. Pract. Exper.** A comparison of approximate string matching algorithms, journal, v.26, n.12, p. 1439–1458, Dec. 1996.

[40] KARP, R. **Reducibility among combinatorial problems**. In: Miller, R.; Thatcher, J., editors, COMPLEXITY OF COMPUTER COMPUTATIONS, p. 85–103. Plenum Press, 1972.

[41] KLEINBERG, J.; TARDOS, E. **Algorithm Design**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[42] KNUTH, D. E. **Mathematics of computation**. Estimating the efficiency of backtrack programs, journal, v.29, n.129, p. 122–136, 1975.

[43] KNUTH, D. E. **SIGACT News**. Big omicron and big omega and big theta, journal, v.8, n.2, p. 18–24, Apr. 1976.

[44] KNUTH, D. E. **The art of computer programming, volume 3: (2nd ed.) sorting and searching**. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[45] KNUTH, D. E. **The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1**, volume Volume 4A: Combinatorial Algorithms, Part 1. 1. ed., Addison-Wesley Professional, 2011.

[46] KNUTH, D. E.; MORRIS, J. H. ; PRATT, V. R. **SIAM J. Comput.** Fast pattern matching in strings, journal, v.6, n.2, p. 323–350, 1977.

[47] KULLMANN, O. **Fundaments of branching heuristics**. In: Biere, A.; Heule, M.; van Maaren, H. ; Walsh, T., editors, HANDBOOK OF SATISFIABILITY, volume 185 of **Frontiers in Artificial Intelligence and Applications**, p. 205–244. IOS Press, 2009.

[48] LAWLER, E. L.; LENSTRA, J. K.; KAN, A. R. ; SHMOYS, D. B. **The traveling salesman problem: a guided tour of combinatorial optimization**, volume 3. Wiley Chichester, 1985.

[49] LEVITIN, A. **Introduction to the Design and Analysis of Algorithms**. Always learning. 3rd. ed., Addison Wesley, 2011.

[50] LIN, T.-C.; LEE, D. T. **Theor. Comput. Sci.** Randomized algorithm for the sum selection problem, journal, v.377, n.1-3, p. 151–156, May 2007.

[51] MANBER, U. **Introduction to Algorithms: A Creative Approach**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[52] MCCREIGHT, E. M. **J. ACM**. A space-economical suffix tree construction algorithm, journal, v.23, n.2, p. 262–272, Apr. 1976.

[53] MICHAEL STEELE, J.; YAO, A. C. **Journal of Algorithms**. Lower bounds for algebraic decision trees, journal, v.3, n.1, p. 1–8, 1982.

[54] MITZENMACHER, M.; UPFAL, E. **Probability and Computing: Randomized Algorithms and Probabilistic Analysis**. New York, NY, USA: Cambridge University Press, 2005.

[55] MOOSA, T. M.; RAHMAN, M. S. **Inf. Process. Lett.** Indexing permutations for binary strings, journal, v.110, n.18-19, p. 795–798, Sept. 2010.

[56] MOOSA, T. M.; RAHMAN, M. S. **J. of Discrete Algorithms**. Subquadratic time and linear space data structures for permutation matching in binary strings, journal, v.10, p. 5–9, Jan. 2012.

[57] PARIDA, L. **Gapped permutation patterns for comparative genomics**. In: ALGORITHMS IN BIOINFORMATICS, volume 4175 of **Lecture Notes in Computer Science**, p. 376–387. Springer Berlin Heidelberg, 2006.

[58] PERUMALLA, K. S.; DEO, N. **Parallel Processing Letters**. Parallel algorithms for maximum subsequence and maximum subarray, journal, v.5, p. 367–373, 1995.

[59] POHL, I. **Commun. ACM**. A sorting problem and its complexity, journal, v.15, n.6, p. 462–464, June 1972.

[60] RABIN, M. O. **Journal of Computer and System Sciences**. Proving Simultaneous Positivity of Linear Forms, journal, v.6, p. 639–650, 1972.

[61] REINGOLD, E. M. **J. ACM**. On the optimality of some set algorithms, journal, v.19, n.4, p. 649–659, Oct. 1972.

[62] RUZZO, W. L.; TOMPA, M. **A linear time algorithm for finding all maximal scoring subsequences.** In: Lengauer, T.; Schneider, R.; Bork, P.; Brutlag, D. L.; Glasgow, J. I.; Mewes, H.-W. ; Zimmer, R., editors, ISMB, p. 234–241. AAAI, 1999.

[63] SEDGEWICK, R.; FLAJOLET, P. **An Introduction to the Analysis of Algorithms**. Pearson Education, 2013.

[64] TAKAOKA, T. **Electr. Notes Theor. Comput. Sci.** Efficient algorithms for the maximum subarray problem by distance matrix multiplication, journal, v.61, p. 191–200, 2002.

[65] TAMAKI, H.; TOKUYAMA, T. **Algorithms for the maximum subarray problem based on matrix multiplication**. In: PROCEEDINGS OF THE NINTH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE

ALGORITHMS, SODA '98, p. 446–452, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[66] TARJAN, R. E. **Data structures and network algorithms**. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1983.

[67] TURING, A. M. **Proceedings of the London Mathematical Society**. On computable numbers, with an application to the entscheidungsproblem, journal, v.42, p. 230–265, 1936.

[68] UKKONEN, E. **Algorithmica**. On-line construction of suffix trees, journal, v.14, n.3, p. 249–260, 1995.

[69] WEINER, P. **Linear pattern matching algorithms**. In: PROCEEDINGS OF THE 14TH ANNUAL SYMPOSIUM ON SWITCHING AND AUTOMATA THEORY (SWAT 1973), SWAT '73, p. 1–11, Washington, DC, USA, 1973. IEEE Computer Society.

[70] WELFORD, B. P. **Technometrics**. Note on a method for calculating corrected sums of squares and products, journal, v.4, n.3, p. pp. 419–420, 1962.

[71] WILLIAMS, V. V. **Breaking the Coppersmith-Winograd barrier**. 2011.