Algorithms and Bounds for Tower of Hanoi Problems on Graphs

Thesis submitted as part of the requirements for the M.Sc. degree of Ben-Gurion University of the Negev

by

Rafi Cohen

The research work for this thesis has been carried out at Ben-Gurion University of the Negev under the supervision of Prof. Yefim Dinitz

May 2008
Subject: Algorithms and Bounds for Tower of Hanoi Problems on Graphs

This thesis is submitted as part of the requirements for the M.Sc. degree

Written by: Rafi Cohen
Advisor: Prof. Yefim Dinitz
Department: Computer Science
Faculty: Natural Sciences
Ben-Gurion University of the Negev

Author signature: ____________ Date: ______
Advisor signature: ____________ Date: ______
Dept. Committee Chairman signature: ____________ Date: ______
Table of Contents

Table of Contents iv

List of Figures vi

Abstract vii

Acknowledgements ix

1 Introduction and History of the Problem 1
  1.1 Origins ................................................. 1
  1.2 Some Definitions .................................. 3
  1.3 Previous Work on Hanoi Problems ................. 5
    1.3.1 Other Versions ............................. 5
    1.3.2 Properties of Solutions ................. 10
  1.4 Our Research Domain .............................. 13
  1.5 Our Results and Structure of the Work .......... 15
  1.6 Directions of Further Research ................ 17

2 Preliminaries 20
  2.1 Definition and Notations .......................... 20
  2.2 Block-Move Properties ............................ 25
  2.3 Basic Results .................................... 28
  2.4 Auxiliary Mathematical Results ................ 29
3 Bounds for Maximal Almost-Acyclic Hanoi Graphs

3.1 Best Almost-Acyclic Hanoi Graph $H_{TC}^\ell$ .................. 35
3.2 Algorithm for $H_{TC}^\ell$ and Its Number of Steps ............... 36
3.3 Lower Bound for a General Algorithm ....................... 38
3.4 Measuring the Tightness of Our Bounds ..................... 41
3.5 Proofs of Theorems 3.2.1(2) and 3.3.1(2) .................... 42
3.5.1 Proof of Theorem 3.2.1(2) ............................... 42
3.5.2 Proof of Theorem 3.3.1(2) ............................... 44

4 Complexity of Parallel-Components Hanoi Graphs

4.1 Exact bounds for $PC_\ell$ ................................. 47
4.2 General Case ........................................ 53
4.2.1 Discrete Solution .................................. 56
4.2.2 Numeric Approximation ................................ 57

5 Bounds for General Almost-Acyclic Hanoi Graphs

5.1 Local Definitions ....................................... 60
5.2 Description of the Algorithm ................................ 61
5.3 Optimality Proof in the Class of Block-Algorithms ........... 67
5.4 General Lower Bound ................................... 69
5.5 Matrix Graph .......................................... 71
5.5.1 Lower Bound .................................. 73
5.5.2 Lower Bound Properties ............................. 75
5.5.3 Algorithm ....................................... 77
5.5.4 Measuring the Tightness of Our Bounds ................ 78

Bibliography 79
List of Figures

1.1  The original 4-peg problem ........................................ 6
1.2  The 5 different strongly-connected-components on 3 vertices 8
2.1  A block-move illustration; $v_s = v_3, \ u_d = u_1$ .............. 24
2.2  The graphs $H_1, H_2$ and $H_2^{\text{max}}$ .......................... 28
3.1  The graph $H_4^{TC}$ .................................................. 35
4.1  Hanoi Graphs $PC_\ell, PC_1$ and other graphs .................... 48
4.2  Parallel Components Hanoi graph $H^{Par}$ ....................... 54
5.1  A example of Hanoi graph with $\ell(G) = 7$. Vertices marked by 61
     a cross may be omitted with no change in the graph of SCCs. 61
5.2  Illustration to Lemma 5.2.2 ....................................... 64
5.3  The $H^{\text{SCC}}$ of the Matrix graph $H = H_{hw}^{\text{Mat}}$ ....... 72
5.4  The parabola $f(\alpha) = \alpha (1 - \frac{\alpha}{2}) \text{ for } 0 \leq \alpha \leq 1$. 76
Abstract

The classic Tower of Hanoi puzzle was marketed by Edouard Lucas in 1883 under the name ”Tower of Hanoi” [31]. There, \( n \) disks are due to be moved from one peg to another, using an auxiliary peg while never placing a bigger disk above a smaller one. Its optimal solution is classic in Computer Science. In this thesis, we study a generalization of the original puzzle.

The problem is generalized by placing pegs on the nodes of a given directed graph (Hanoi graph) \( H = (V(H), E(H)) \) with two distinguished nodes \( S \) and \( D \), and allowing moves only along edges of that path. An optimal solution for \( H \) is a shortest sequence of disk moves to transfer \( n \) disks from \( S \) to \( D \); we denote its length by \( time(H) \). Leiss [29] has shown that arbitrary many disks can be moved from \( S \) to \( D \) if and only if there is a strongly-connected-component (SCC) of size three or more in \( H \), which can be reached from \( S \), and from which \( D \) can be reached. D. Azriel [5] studied Hanoi graphs, with each SCC of size at most two, showing the number of disks that can be moved from \( S \) to \( D \) in those graphs is at most \( h^{\frac{1}{2} \log_2 h} \), where \( h = |V(H)| \). Recently, Hanoi graphs consisting of (a SCC of size three, a path from \( S \) to one of its nodes and an edge from \( S \) to that node) were studied, an algorithm was suggested, and its optimality was proved [7].

We consider graphs which are composed of \( \ell \) SCCs, each one isomorphic to \( K_3 \), and concentrate on the logarithm of \( time(H) \), looking for asymptotic
bounds for $\log_2(\text{time}(H))$ with respect to $n$, with $\ell$ as a parameter.

Our algorithms are based on a fixed, but not given a-priori, division of the $n$ disks to blocks of consecutive disks, and moving those blocks during the algorithm, block by block. The blocks are (almost) equal sized. We call such algorithms block-algorithms.

Our main mathematical technique is establishing recurrences for the upper and lower bounds for the expression $\frac{n}{\log(\text{time}(H))}$. We then advance in the two following (independent) directions aiming in showing that our bounds are tight. First, we give analytical forms for the recurrences as above. Second, we estimate the gap between the upper and lower bounds directly using the recurrences.

We start with defining and analyzing the most connected Hanoi graph with $\ell$ SCCs, which minimizes $\text{time}(H)$. We prove a tight bound for its number of steps which is approximately $2^{f(\ell)n}$, where $f(\ell) \approx \ell \frac{1}{2} \log \ell$. Then we proceed to the least connected Hanoi graph with $\ell$ SCCs, which maximizes $\text{time}(H)$. We show the number of steps to be roughly $2^{n \ell}$. Next, we provide an algorithm and a lower bound for general Hanoi graphs, in a form similar to that for the most connected Hanoi graphs. We conclude in studying Hanoi graphs with a topology resembling a matrix, which are in a sense a ”middle-way” between the most and least connected Hanoi graphs. We prove a lower bound of approximately $w^{\log h} + 1 + h^{\frac{1}{2} \log h}$, and an upper bound of approximately $wh(\frac{1}{2} - o(1) \log h)$, where $w$ and $h$ are the width and height of the matrix respectively. We also show the recurrences defining the bounds to be tight.
Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Yefim Dinitz. During the thesis, I was always welcome, and each meeting with him contributed a lot to the advance of the work. Many of the ideas grew out of our conversations, and many improvements were based upon his insights. He is an advisor in the full sense of the word.

I wish to thank Prof. Daniel Berend for his helpful suggestions.

I would also like to thank Shay Solomon, and Amir Sapir for all the help and suggestions they provided.

I am especially grateful to Natali, for her unconditional love and support. Lastly, I would like to thank my parents, for their affection, love and help, throughout the years.
Chapter 1

Introduction and History of the Problem

1.1 Origins

The original, well studied problem of the "Towers of Hanoi" invented in 1883 by Lucas [31] can be stated as follows. There are three pegs, $S$, $D$, and $A$. Peg $S$ (source) contains $n$ disks, no two of them of equal size, and upon any given disk only smaller disk may rest. The problem is to find the shortest sequence of steps, to move the $n$ disks from $S$ to $D$ (destination) using peg $A$ as auxiliary peg, subject to the following three rules:

(R1) Only one disk may be moved at a time; a move consists of transferring the topmost disk from one peg to the top of all the disks on another peg.

(R2) A disk may be moved from any peg to any other peg.

(R3) At no time may a disk be placed upon a smaller disk.
We refer later to constraints 1-3 as the *Hanoi rules* (HR).

The second peg, which is seemingly unimportant as far as the formulation of the problem goes, is what really enables us solving it. In fact, for \( n \geq 2 \) the problem would be unsolvable if we had only the source and destination pegs.

Initially, the problem was posed both as a game and as a legend. As a game it was built of a plate, three wooden sticks (the pegs) and several (originally 8) disks of mutually distinct diameters. As a legend, a story was told, aimed at demonstrating the fast increase in size of an exponential expression. There, the number of disks was 64. As we shall see momentarily, with this number of disks, the number of steps required for the solution is over \( 10^{19} \).

The shortest solution of the problem requires exactly \( 2^n - 1 \) steps [31].

The well-known recursive solution is presented in Algorithm 1.1.

**Algorithm 1.1** Recursive3peg\((n, S, D, A)\)

1: if \( n \geq 1 \) then
2:    Recursive3peg\((n - 1, S, A, D)\).
3:    move disk number \( n \) from \( S \) to \( D \)
4:    Recursive3peg\((n - 1, A, D, S)\).
5: end if

The correctness of the algorithm is straightforward, and so is the running time analysis, which goes as follows. Let \( a \) denote the number of moves required to perform the task with \( n \) disks. Clearly \( a_1 = 1 \) and \( a_n = 2a_{n-1} + 1 \), yielding \( a_n = 2^{n-1} \). This solution is optimal, as was discussed in [8], and
several others (cf. [43, 34]).

Even though the solution seems simple enough, there has been continuing interest in the problem from several points of view. Considered as a game, many programs were written, demonstrating the solution by animation. For educational purposes, it serves in introductory programming courses as perhaps the best example of a recursive program. In fact, as opposed to many other recursive procedures, here it is much more elaborate to present an iterative solution, and in particular prove its correctness.

There has also been a great interest in various generalizations of the problem, and in various properties of optimal solutions.

Before turning to the main part of the work, we present notations in Section 1.2 and review work done previously in Section 1.3. Section 1.4 states the main questions studied in the current work and the main results. The structure of the work is described in Section 1.5.

1.2 Some Definitions

A configuration is any arrangement of the disks among the pegs, in accordance with HR. A perfect configuration is one in which all disks are on the same peg. A perfect task is the transfer of a tower of $n$ disks from a perfect configuration in $S$ to a perfect configuration in $D$. Let $H$ be a Hanoi graph. For each $n$, we have a configuration graph $H^{(n)}$, whose vertices are all configurations of $n$ disks, and there is an edge from vertex $v_i$ to $v_j$ if one can
pass from the former to the latter by a single move.

A *Hanoi graph* is a simple, directed, finite graph $H = (V, E)$ with two distinguished vertices, denoted by $S$ and $D$, $S \neq D$, such that (without loss of generality) for each vertex $v \in V$ there is a path from $S$ to $v$ and a path from $v$ to $D$. At each vertex of $H$ there is a peg, which is identified with the vertex itself, the number of pegs is $h$. The source initially contains $n$ disks of different sizes, such that smaller disks rest on top of larger ones. HR apply, with the following restriction:

(R2’) A disk may be moved from a peg $v$ to another peg $w$ only if there is an edge from $v$ to $w$, i.e. $(v, w) \in E$.

The Tower of Hanoi problem $\text{HAN}(H, n)$ for a Hanoi graph $H$ and $n \geq 0$ is to transfer $n$ disks of distinct sizes from $S$ to $D$, subject to the HR.

A problem $\text{HAN}(H, n)$ is *solvable* if the aforementioned task may be accomplished. A Hanoi graph $H$ is a *solvable* graph if $\text{HAN}(H, n)$ is such for all $n \geq 0$.

The requirement that for each vertex $v \in V$ there is a path from $S$ to $v$ and a path from $v$ to $D$ is designed to get rid of inessential vertices. In fact, if there is no path from $S$ to $v$ or from $v$ to $D$, then no solution to $\text{HAN}(H, n)$ may use the peg $v$, so that $v$ may be omitted from the graph.

Leiss [29] obtained the following characterization of solvable Hanoi graphs. (Note that, in his formulation of the problem, $V$ may contain inessential vertices, and hence the formulation of the theorem hereby given is changed
Accordingly.)

**Theorem 1.2.1** [29] Let $H = (V, E)$ be a Hanoi graph, and let $H^* = (V, E^*)$ be its transitive closure: $E^* = \{(v, w) : v \neq w, \text{there is a path from } v \text{ to } w\}$. Then $H$ is solvable if and only if $H^*$ contains a clique of size three.

### 1.3 Previous Work on Hanoi Problems

#### 1.3.1 Other Versions

Since the time of its inception, people have started developing versions of the original puzzle, such as starting and ending with any configuration, adding pegs, and imposing movement restrictions among the pegs, to mention just a few. Here we will give a short description of some most known of them. Each version will be titled relative to the well-known 3-peg problem. A comprehensive list of generalizations of the problem is mentioned by P.K. Stockmeyer [41].

**Adding Pegs**

One of the known early versions of this natural extension is the so-called "Reve’s Puzzle" [16]. There, a limited form of 4 pegs and a choice of 8, 10 or 21 disks was presented (see Fig. 1.1), and one is required to move the tower of disks from peg 1 to peg 4. The problem in its general form, that is any number of $h \geq 4$ pegs and any number $n$ of disks, was presented in [39], in two papers. Algorithm 1.2 depicts a solution for 4-peg versions.

In fact, the algorithm is not clear, as $m$ is left unspecified. It has to be
Figure 1.1: The original 4-peg problem

Algorithm 1.2 FrameStewart4peg(1,...,n, S, D, A1, A2)
1: if n ≥ 1 then
2:   for some m do
3:     FrameStewart4peg(1,...,n−m, S, A2, A1, D).
4:     move the largest m disks from S to D using A1 as the spare peg
5:     FrameStewart4peg(1,...,n−m, A2, D, A1, S).
6:   end for
7: end if
chosen so that the overall number of moves will be minimal. It is well known that, for 4 pegs, $m$ should be approximately $\sqrt{2n}$. A variety of algorithms have been proposed in the past few decades. Recently, [25] discussed several of them and proved that they are essentially the same.

As was noted in [17], the solutions proposed by Stewart and Frame lacked an optimality proof. Their assumption, whereby considering all possible values for $m$ we obtain a minimal solution, became later known as the "Frame-Stewart conjecture", and is still open. Knuth expressed his opinion on the conjecture, saying "I doubt if anyone will ever resolve the conjecture; it is truly difficult." (cf. [32]). Employing exhaustive search, it has been verified in [10] that, up to 17 disks, the Frame-Stewart algorithm yields indeed an optimal solution. Using ‘economic’ search techniques, Korf [28] performed exhaustive search of up to 30 disks, which still agrees with the numbers obtained by Frame-Stewart’s algorithm.

A time analysis of the algorithm yields, that the growth rate of the number of moves is bounded by the sub-exponential expression $O(n^{h-3}2^{n^{1-h}})$ [40]. The lower bound issue has been discussed in [42], where a sub-exponential bound of $2^{\frac{1}{2}(1+o(1))(\frac{12}{\pi h-n})\frac{n}{h-2}}$ was obtained. Recently, that bound has been improved to $2^{(1+o(1))(h-2)ln}\frac{n}{h-2}$ in [13].

**Imposing Movement Restrictions Among the Pegs**

Several researchers considered Hanoi graphs in which disk moves can be performed only along the edges. The earliest known version is the three-in-a-row,
Figure 1.2: The 5 different strongly-connected-components on 3 vertices described in [38] and [40], where there are pegs 1, 2, 3 and peg 1 is connected to peg 2 bi-directionally, the same between pegs 2 and 3, but no edges between pegs 1 and 3.

In [4], the Cyclic version, in which the pegs are ordered in a circle, and at each move we are allowed to move a disk from one peg to another only in the counter-clockwise direction, was studied.

Optimal solutions for the five non-isomorphic solvable graphs on three vertices are discussed thoroughly in [37], see Figure 1.2. The same author presents asymptotic bounds for the number of moves in the Cyclic version for more than 3 pegs in [9].

Recently, Hanoi Graphs consisting of a strongly-connected-component (SCC) of size three, a path to one of its nodes and an edge from S to that node were studied, an algorithm was suggested, and its optimality proved
Any Initial and Final Configurations

The problem of starting from any specified legal configuration and ending with another specified one was suggested in [18]. A nice recursive algorithm to solve it is described in [20].

Versions which are not Strongly Connected

Given a Hanoi graph, a question may arise: how many disks can be moved from source to destination? Theorem 1.2.1 states the necessary and sufficient condition for the existence of a solution. For Hanoi graphs not satisfying the condition in the Theorem, the number of disks has been shown to be maximally of the order of \( h^{1/\log_2 h} \) in [6]. The same paper also gives a characterization of a class of graphs yielding that order of number of disks (cf. [30]).

Allowing a Disk on Top of a Smaller One

There are two versions:

a) Allowing a disk on top of a smaller one at the initial arrangement.

Here, one is allowed to let this situation stay on as long as it is convenient, but not to create more situations like this in the course of solution. The number of moves is bounded from above by \( 2^n + 2^{n-2} - 1 \) (see [26]).
b) Allowing a disk to be placed on top of a smaller one provided that the difference in size between the two does not exceed a given $k$.

This issue has been dealt with by several researchers (cf. [15]).

Assigning Colors to Disks

Coloring the disks, we obtain the "Towers of Antwerpen" problem, where on each peg a tower of disks of a certain color is placed, and the task is to replace the pegs on which the disks reside. In [33], the number of moves the optimal procedure requires was found to be $3 \cdot 2^{n+2} - 8n - 10$. A version consisting of two towers of disks has been proposed and solved in [40]. The same author also proposed the Rainbow problem [38], in which there is just one tower of disks of several colors, and at no stage are two disks of the same color allowed to be adjacent.

1.3.2 Properties of Solutions

Perhaps one of the most elegant properties of instances of solutions to the original "Towers of Hanoi" problem is the similarity to the common Gray code: the sequence of bit numbers that change, when moving from a codeword to its successor, is identical to the sequence composed of the numbers of the disks moving at each stage in an optimal solution [21]. The following subsections will discuss several other properties of instances of solutions.
Analogies to Other Problems

The configuration graph for some $n$ resembles several other structures. By writing down the Pascal Triangle with $2^n$ rows, deleting the even numbers, and drawing edges between neighbouring numbers, one obtains the structure of the Hanoi configuration graph for $n$ disks. This and more analogies, together with analysis of their properties, can be found in [24]. Another example is given in [14], where it is shown that a Hamiltonian tour through the vertices of the $n$-dimensional cube is analogous to the chain of states in the optimal solution of the Tower of Hanoi.

Quantitative Properties of Solutions

Allowing any source configuration and any destination configuration, researchers were interested in finding the average distance between any two configurations ([19, 12, 24]). Another question is "what is the minimal number of moves the largest disk has to make in an optimal solution?" It receives a non-trivial answer in [24], where it is shown that (unlike 3-pegs perfect tasks), being greedy with respect to the largest disk, we may be led to a non-optimal solution. The complexity of finding out whether the largest disk should be moved once or twice was resolved in [36].

Iterative Algorithms

As was mentioned in Section 1.1, composing an iterative solution for (a version of) this problem is much more difficult. Besides being a programming
challenge, an iterative algorithm may shed light on some properties of a solution. For example, the amount of moves each disk performs in the classical Hanoi version is much clearer when observing an iterative solution than when looking at the recursive one. In [35], a 'mechanical’ way of transforming the recursive procedure into an iterative one, based on carefully examining the needed pieces of information, and removing the redundant ones, is suggested. Other researchers (cf. [22]; [11]) also discussed the iterative solution to the general problem.

Sequences of Moves

A solution can be represented as a string, where each symbol designates a move, and moves are distinguished according to their pair of source-destination pegs. In ([1]) it is shown that a string which represents an optimal solution is square-free (We say a string $u$ is square-free if it has no factors of the form $yy$, where $y$ is a finite, nonempty string). Using some rules, it is possible to generate an infinite sequence such that the optimal solution to the classical version, for any number of disks, appears as a prefix of that sequence [2]. This has been extended to several other 3-peg versions [3].

Diameter of Configuration Graphs

Intuitively, a perfect task is a candidate to require the largest number of moves. In [18] and [23] it has been shown that the perfect task of the classical
problem requires more moves than any task between non-perfect configurations. However, this is not always the case. Consider $K_4$ and 15 disks, where $K_4$ is the complete graph on 4 vertices. A perfect task requires 129 moves, whereas Korf [27] found non-perfect configurations that require 130 moves to reach a perfect configuration.

1.4 Our Research Domain

It is well known that if we shrink in a graph $H$, each strongly-connected-component (or in short SCC) to a vertex, we get the graph $H^{SCC} = (V^{SCC}, E^{SCC})$, which is a DAG (directed acyclic graph). We define the following notation:

**Definition 1.4.1** A solvable Hanoi Graph $H = (V, E)$ is called almost-acyclic of order $\alpha$, if the size of each its strongly-connected-component is at most $\alpha$.

We write in short AAG$\alpha$, to denote almost-acyclic Hanoi graph of order $\alpha$. We define $\ell(H)$ as the number of SCCs in $H$, and write AAG$\alpha_{\ell}$ to denote AAG$\alpha$ with $\ell$ SCCs.

D. Azriel [5] studied AAG2. Solvable graphs begin from AAG3. So, we concentrate on almost acyclic of order three Hanoi graphs, focusing on graphs in which each SCC is $K_3$. Since no similar research is known to be done, we consider such a restriction be a reasonable first step to more general Hanoi graph investigation.

Proving exact bounds, i.e., showing an algorithm and proving a matching lower-bound, for an arbitrary almost-acyclic Hanoi graph, looks currently
too difficult. For example, in [7] there is a complicated proof for the exact bound for a very particular case. Also in [5] there are non-matching lower and upper bounds for AAG2. We concentrate on the following:

(i) Suggesting algorithms of length $\approx 2^{n/\ell}$, where $n$ is the number of disks.

(ii) Proving asymptotic lower-bounds for $g(n)$, where the minimal number of moves is $2^{n/\ell}$.

(iii) The bounds in (i) and (ii) are defined by recurrences, so we advance in the two following (independent) directions:

- Trying to give analytical formulas for a lower bound for $f(\ell)$ and for an upper bound for $g(\ell)$.
- Trying to estimate the gap between $f(\ell)$ and $g(\ell)$ by using the recurrences.

In other words, we look for upper and lower bounds for the logarithm of the minimal number of steps (to be more precise, for $\log_2(time)$). We look for asymptotic bounds with respect to $n$, with $\ell$ as a parameter.

In this thesis, we use a special variant $O_\ell(\cdot)$ of the $O(\cdot)$ notation. For a given function $g(n)$, we say that $f(n) = O_\ell(g(n))$ if there exist a function $c(\ell)$, depending on $\ell$ only, and a constant $n_0$ such that $f(n) \leq c(\ell) \cdot g(n)$ for all $n \geq n_0$.

Our algorithms are of a quite restricted class. They are based on a fixed, but not given a-priori, division of the $n$ disks to blocks of consecutive disks,
and moving those blocks during the algorithm, block by block. The blocks are (almost) equal sized. We call such algorithms block-algorithms. It is proven that the logarithm of the running time of a block-algorithm is inversely proportional to the number of blocks it uses.

There is some similarity between the work of D. Azriel on AAG2 [5], and our research. One might think of a disk in the model of [5] as the counterpart of a block of disks in our model. Likewise, our "best" Hanoi graph $H^{TC}_\ell$ and the algorithm for it are similar to those of [5].

In our algorithm for general AAG3, we were surprisingly able to generalize the idea of first moving the disks to the "middle" node, which was quite natural in [5].

We generalized techniques of building recurrences and proving bounds of [5] for general AAG3. We note that direct generalizations of the lower bound techniques of [5] were applicable only for block-algorithms. It was the insight of Proposition 2.4.3, which enabled us to extend our lower-bounds for general algorithms.

1.5 Our Results and Structure of the Work

In Chapter 2 we provide elaborate definitions and some basic results.

In Chapter 3 we deal with the most connected AAG3 $H^{TC}_\ell$, which provide the minimal number of steps among all AAG3$_\ell$. We suggest a block-algorithm $Move_{tc}$ for them. We also provide a recurrence describing its running time,
and give an explicit expression bounding the function defined by it. It implies
a running time of $O(2^{\frac{n}{\ell}})$, where for each $\varepsilon > 0$ there exists a constant
$C_\varepsilon > 0$ s.t. $f(\ell) \geq C_\varepsilon \ell^{(\frac{1}{2}-\varepsilon)\log \ell}$ for each $\ell$.

On the other hand, we provide a recurrence for a lower bound, and give
an explicit expression bounding the function defined by it. It implies a lower-
bound of $\Omega(\frac{n}{\ell})$, where there exists a constant $C$ such that $g(\ell) \leq C\ell^{\frac{1}{2}\log \ell}$
for each $\ell$. Finally, by studying the relation between the two recurrences, we
prove the upper bound of $6 \cdot \ell^{\log_2 3} < 6 \cdot \ell^{1.6}$ for the ratio $\frac{g(\ell)}{f(\ell)}$, which is of a
lesser order than the order of magnitude of $\frac{C\ell^{\frac{1}{2}\log \ell}}{C_\varepsilon \ell^{(\frac{1}{2}-\varepsilon)\log \ell}}$.

In Chapter 4 we study a family of graphs, which are the least connected
among AAG$_3$, with the maximal number of steps in the optimal algorithm.
This family consists of the so called ”parallel-components” Hanoi graphs
PC$_\ell$. Each PC$_\ell$ is composed of $\ell$ $K_3$, where for each component there is an
edge from $S$, and another edge to $D$. We provide an optimal algorithm for
this family, which requires $(\ell + r)2^{\frac{\ell}{2}} \leq \ell \cdot 2^{\frac{\ell}{2}}$ steps, $r \equiv n \pmod \ell$. At
the end of this section, we generalize our results to Hanoi graphs ”parallelly-
composed” of arbitrary Hanoi graphs, instead of $K_3$.

In Chapter 5 we provide the algorithm Move$_{\text{gen}}$ for a restricted class
of general AAG3. The restriction is that it is possible to omit from each
cOMPONENT a vertex without effecting $H^{\text{SCC}}$, or more formally:

$$\forall C \in V(H^{\text{SCC}}) \exists v_c \in C \text{ s.t., } \tilde{H}^{\text{SCC}} = H^{\text{SCC}}, \text{where } \tilde{H} = H \setminus \{v_c\}_{c \in V(H^{\text{SCC}})}.$$
We provide recurrences for upper and lower bounds, which have a form similar to that in Chapter 3. We were unable to estimate the gap between the lower and upper bound, in the general case. Although, we believe them to be tight, similarly to the tightness proved in Chapter 3.

We use the results of the beginning of this chapter to study a family of graphs called "Matrix" Hanoi graph $H_{hw}^{Mat}$, which has a topology resembling a matrix with $h$ rows and $w$ columns. Matrix graphs are in a sense the "middle way", between "row-graphs" $PC_{\ell}$ and "column-graphs" $H_{Tc}^{TC}$. We consider them to be a kind of approximation for the general case.

Applying the aforementioned results, we got an upper bound of $O_{\ell}(2^{\frac{w}{\log h}})$, where for each $\varepsilon > 0$ there exists a constant $C_\varepsilon > 0$, such that $f(h, w) \geq C_\varepsilon \cdot w \cdot h^{(\frac{1}{2} - \varepsilon) \log h}$ for each $h \geq 1$. On the other hand, we got a lower bound of $\Omega_{\ell}(2^{\frac{w}{\log h}})$, where there exists a constant $C$, such that $g(h, w) \leq Cw^{[\log h]+1}h^{\frac{1}{2} \log h}$ for any $w, h \geq 1$. Finally, by studying the relation between the two recurrences, we prove the upper bound of $6 \cdot h^{\log_2 3} < 6 \cdot h^{1.6}$ for the ratio $\frac{g(h, w)}{f(h, w)}$, which is of a lesser order than the order of magnitude of $\frac{Cw^{[\log h]+1}h^{\frac{1}{2} \log h}}{C_\varepsilon \cdot w^{[\log h]}h^{\varepsilon \log h}}$.

$1.6$ Directions of Further Research

In our work we focus on Hanoi graphs built from complete SCCs of size three, i.e., $K_3$. Note that all graphs with a single SCC of size three are studied by A. Sapir [37]. We see it interesting to study the following problems, generalizing
those studied in this work.

1. Adding unsolvable SCCs, i.e., components of size one or two. We checked that it is possible to add $\ell$ such components to $H^{TC}_\ell$ between the original components, so that the number of blocks in the algorithm increases by a constant factor. A question arises, what may be the influence of the addition of unsolvable SCCs in general.

2. Allowing components of size three, which are not $K_3$.

(a) All SCCs are of the same kind. We believe our approach could be (easily) adjusted to those graphs, yielding similar results.

(b) There are SCCs of different kind. In [37], it is proven that the running time for any SCCs of size three is $C\lambda^n$, where $2 \leq \lambda \leq 3$. We believe our approach could be adjusted to those graphs, so that the logarithm of running time is multiplied by $\log \lambda$.

3. Allowing components of size $\geq 4$, i.e., $K_4$ (Frame-Stewart), cycles, etc.

(a) All SCCs are of the same kind. Our believe is similar to that in item 2a.

(b) There are SCCs of different kind. We can define the relation " $\prec$ " for SCCs in $H$ as follows: $C' \prec C''$ iff $\log(\text{time}(C')) = o(\log(\text{time}(C''))).$ For each pair of SCCs, $C'$ and $C''$, s.t., $C' \prec C''$, we shrink $C'$ to a vertex. We believe that the graph $H'$ obtained
from such shrinkage, satisfies \( \log(\text{time}(H')) = O(\log(\text{time}(H))) \).

It might be possible to apply our techniques on \( H' \).

4. What may be the influence of canceling the restriction that we use for general graphs (see Section 1.5) on the algorithm?

5. We were able to estimate the gap between the lower and upper bounds for \( H^{TC} \) and \( H^{Mat} \). We conjecture the gap in the general case to be \( \frac{g(H)}{f(H)} \leq C \cdot 3^{D(H)} \), where \( C \) is a constant and \( D(H) \) is the depth of the recurrence for computing the lower-bound for \( H \) (which is true for the cases of \( H^{TC}_t \) and \( H^{Mat}_t \)).

6. Small-but-interesting questions:

   (a) Our general algorithm \( Move_{gen} \) and general lower-bound are based on the topology of \( H^{SCC} \). How would addition of edges to \( H \), while keeping the same \( H^{SCC} \), influence the results?

   (b) Generalization of block moves, and its influence on the running time of our block-algorithms. Such as using multiple move-paths for a single block move.

   (c) Finding exact solutions for small Hanoi graphs, such as a graph composed of two \( K_3 \) connected by an edge.
Chapter 2

Preliminaries

2.1 Definition and Notations

In this paper, a disk set \( D \) is a subset of \( [1, \ldots, n] \). A configuration of a disk set \( D \) is a specification of disjoint ordered sets of disks on the pegs, in accordance with HR, whose union is \( D \). A perfect configuration is one in which all disks reside on the same peg.

We say that \( D \) is on a vertex \( v \), if all the disks of \( D \) are on the peg identified with \( v \). We say \( D \) is in \( C \), where \( C \) is a component in the graph, if all the disks of \( D \) reside on the vertices belonging to \( C \). \( D \) is in perfect-configuration in \( C \), if \( D \) is in perfect-configuration in one of the vertices belonging to \( C \).

A move of disk \( r \) from peg \( u \) to peg \( v \) is denoted by \( u \stackrel{r}{\rightarrow} v \). Obviously, a configuration of \( D \setminus \{r\} \) is the same before and after such a move; we refer to it as the configuration of \( D \setminus \{r\} \) during \( u \stackrel{r}{\rightarrow} v \).

The length of a sequence of moves \( P \) is the number of moves in it, denoted
by $|P|$. Given a sequence of disk moves $P$, containing at some point the move $u \xrightarrow{r} v$ and at some later point the move $u' \xrightarrow{r'} v'$, the subsequence of $P$ of all moves between these moves, including the former move but not the later one is denoted by $P[u \xrightarrow{r} v, u' \xrightarrow{r'} v']$, or simply $[u \xrightarrow{r} v, u' \xrightarrow{r'} v']$ if the move sequence is clear from the context. Similarly $P(u \xrightarrow{r} v, u' \xrightarrow{r'} v')$ denotes the subsequence of $P$ between these moves, excluding the former and the later moves. In general we may use also open-closed and closed-closed intervals.

For any configuration $C$ and disk set $D$, let us define the restriction of $C$ to $D$, denoted by $C|_D$, as the configuration $C$ with all the disks not in $D$ removed. Similarly, for any sequence of moves $P$ of $D$ from a configuration $C_1$ to another configuration $C_2$, and any disk subset $D'$, we define the restriction of $P$ to $D'$, denoted by $P|_{D'}$, as the result of omitting of all moves of disks not in $D'$ from $P$; clearly, $P|_{D'}$ transforms $C_1|_{D'}$ to $C_2|_{D'}$. It is easy to see that the restriction of any legal sequence of moves to any disk subset is legal. Thus, for any two disk sets $D' \subset D$ and any move sequence $P$ of $D$, the restriction $P|_{D'}$ is a legal move sequence of $D'$. Moreover, the restriction $P|_{D\setminus D'}$ is a legal move sequence of $D\setminus D'$, and $|(P|_{D\setminus D'})| + |(P|_{D'})| = |P|$ holds.

We say a Hanoi graph is empty, if all disks reside in a perfect-configuration either in $S$, or in $D$. Suppose that the transfer of a disk set $D_i$, $i \in \{1, \ldots, k\}$, from $S_i$ to $D_i$ in an empty graph requires $t_i$ steps. Consider the task of transferring $D = \bigcup_{i=1}^k D_i$, in an empty graph, where in the initial configuration each disk set $D_i$ reside on $S_i$, and in the final configuration each $D_i$ reside
on \( D_i \). Observe that any solution \( \mathcal{S} \) of such a task requires at least \( \sum_{i=1}^{k} t_i \) steps, since for any \( i \), the restriction \( \mathcal{S}|_{D_i} \) contains at least \( t_i \) steps.

The *composition* of several sequences of moves \( P_i \) \( (i = 1, \ldots, t) \) of the same disk set \( \mathcal{S} \) is defined as the concatenation of their related sequences of moves. If the final configuration of each \( P_i, i \neq t \), coincides with the initial configuration of \( P_{i+1} \), then such a composition is legal.

Given a sequence of moves \( P \), the inverse sequence, denoted by \( \overline{P} \), is the sequence in which the \( i \)'th move of \( \overline{P} \) is the \( (|P| + 1 - i) \)'th move in \( P \) (e.g. the first of move \( P \) is the last move of \( P \)). Note that \( |P| = |\overline{P}| \). Moves in \( \overline{P} \) are legal for \( \overline{H} \), where \( \overline{H} \) is obtained from \( H \) by reversing the direction of all edges of \( H \). So in case \( H = \overline{H} \), the composition of \( P \) and \( \overline{P} \) is legal, resulting in the initial configuration before activating \( P \).

The almost-acyclic Hanoi graphs \( H \) studied in this thesis contain \( \ell \) disjoint graphs, \( C(H) = \{C_1, C_2, \ldots, C_\ell\} \), of size three each. We denote the vertices of \( C_i \) by \( \{v_{i1}, v_{i2}, v_{i3}\} \). We assume that \( C(H) \) is numbered w.r.t. some topological sort of \( H^{SCC} \), i.e., if there’s a directed path from \( C_i \) to \( C_j \) in \( H^{SCC} \), then \( i < j \). We also assume that \( S = v_{11} \) and \( D = v_{\ell 1} \).

**Definition 2.1.1** For two components \( C_i, C_j \in C(H) \), we define Hanoi graph \( H_{ij} = H[C_i, C_j] \):

\[
V(H_{ij}) = \{v : \text{There is a path from } v_{i1} \text{ to } v \text{ and a path from } v \text{ to } v_{j1}\}.
\]

\[
E(H_{ij}) = \{(u, v) : (u, v) \in E(H) \text{ and } u, v \in V(H_{ij})\}.
\]

\[
\text{Source}(H_{ij}) = v_{i1}, \text{Destination}(H_{ij}) = v_{j1}.
\]
We use the notation $I(H_{ij})$ to indicate the set of indices of the components in $C(H_{ij})$.

Given the number of disks $n$ and another parameter $k$, for simplicity reasons we assume $k|n$. We define the following $k$ blocks: $B_1 = [1..\frac{n}{k}], B_2 = [(\frac{n}{k} + 1)..2\frac{n}{k}], \ldots, B_k = [(k - 1)\frac{n}{k} + 1)..n]$. We sometimes identify a block $B_i$ with its index, e.g., for two blocks $B_i$ and $B_j$, we say that $B_i > B_j$, if $i > j$.

For the following definition, it will be convenient to encode legal sequences of moves of disks by paths of $H$. Thus $(m, p)$, where $p$ is the path $(e_1, e_2, \ldots, e_t), e_i \in E$, signifies that disk $m$ first moves from the starting point of $e_1$ to the endpoint of $e_1$, then it moves from starting point of $e_2$ to the endpoint of $e_2$, and so forth, up to the end point of $e_t$.

In the construction of algorithms in this thesis, we deal with special sequences of disk moves, called block-moves. Let $C_i$ and $C_j$, be two complete SCCs in $H$, with the vertices $\{v_1, v_2, v_3\}$, and $\{u_1, u_2, u_3\}$ correspondingly. We assume there is a path, $p$, between them, and w.l.o.g $p$ is from $v_2$ to $u_2$. For those components and some block $B$, a block-move is defined as follows:

**Definition 2.1.2** Let $C_i$ and $C_j$ be some components in $H$, with a path between them, $p$, and $B$ some block. A sequence of moves $BM$ is called a block-move of block $B$ from $C_i$ to $C_j$, if it transfers legally all disks that belong to $B$ from $C_i$ to $C_j$ with the following restrictions:

1. In the initial configuration, there exists a peg in $C_i$, where $B$ resides on top of it; let us denote it by $v_s$. The same apply for the final configuration and a peg in $C_j$; let us denote it by $u_d$.
2. $BM$ only moves disks that belong to $B$.
3. $BM$ is one of the following three types:
(a) In the degenerate case where $C_i = C_j$, BM transfers all disks in $B$ from $v_s$ in $C_i$ to $v_d$ in $C_i$, using Recursive3peg (Algorithm 1.1).

(b) BM is a simulation of Recursive3peg with the destination $C_j$ instead of $C_i$ in the following way: Execute Recursive3peg from $v_s$ to $v_2$ with each last move of disk $m$ in the algorithm, $(m, (v_x, v_2))$, replaced with the sequence $(m, ((v_x, v_2) \cup p \cup (u_2, u_d)))$.

(c) This type is a mirror-image of type (3b). We execute Recursive3peg from $u_2$ to $u_d$, this time with the first move of disk $m$, $(m, (u_2, u_x))$, replaced with the sequence $(m, ((v_s, v_2) \cup p \cup (u_2, u_x)))$.

It is easy to see that any block-move is legal, see Figure 2.1 for a block-move illustration.

We call the path used to transfer the disks between the two components a move-path. In block-moves of type (3a) and (3b) (henceforth $BM_{(a)}$ and $BM_{(b)}$ accordingly), we say $C_i$ is being used as the sorting-component, while in (3c) (henceforth $BM_{(c)}$), $C_j$ is being used as the sorting-component.

A block-move was defined here for two complete components, though it could be generalized to any two components of size three, using Sapir’s Unified algorithm [37], instead of Recursive3peg algorithm.
With the definition of a block-move, we can define the notion of a block-algorithm.

**Definition 2.1.3** A block-algorithm is a composition of block-moves, assumed that the number of blocks, and thus the blocks themselves are fixed along the algorithm.

### 2.2 Block-Move Properties

From the definition of a block-algorithm we notice the following fact:

**Observation 2.2.1** In a block-algorithm, at the initial configuration and after any block-move all vertices contain complete blocks.

Thus, we can refer to block-moves as atomic-actions, and use the notation defined in the previous section for them, e.g., using the interval notation and encoding a sequence of block-moves by paths of $H$. We can also say in short, that we transferred a block $B$, from some vertex $v$, to some component $C$, to mean we transferred $B$ from $v$ to some vertex in $C$, by a block-move.

The following condition is necessary and sufficient for a block-move of $B$ from $C_i$ to $C_j$ to be possible.

**Observation 2.2.2** For a block-move of $B$ from $C_i$ to $C_j$ by a move-path $p$ to be possible, it is necessary and sufficient that during it, at least one component out of $C_i$ and $C_j$, as well as all vertices in the move-path and $v_s$ and $v_d$ are free of disks from blocks smaller than $B$.

**Proof:** If the condition in the observation is met, than we can use a block-move of the type that uses the guaranteed free component as the sorting-component.
In case the condition is not met, then we have two cases:

1. The move-path p is not empty of small disks.

2. Both components are not free of them.

In case 1, we cannot use the move-path p to transfer disks belonging to B, while restriction 2 forbids to remove small disks from p. In case 2, according to the second restriction, any block-move transfers only disks which belong to B. Therefore, it uses at most two vertices in each of the components \( C_i \) and \( C_j \). The subgraph induced by those vertices and the move-path is unsolvable due to Theorem 1.2.1. In other words, we can transfer only a finite amount of disks from \( C_i \) to \( C_j \), a contradiction. \( \square \)

**Proposition 2.2.3** The cost of a block-move of a block \( B \) using move path \( p \) is \( 2^m + m \cdot (|p| + 1) - 1 \).

**Proof:** A block-move consists of an application of Recursive3peg and a transfer of each disk along \( p \). The cost of Recursive3peg is \( 2^m - 1 \), while the cost of the latter transfer is \( m \cdot (|p| + 1) \). Hence, the proposition follows. \( \square \)

In this thesis, all our block-algorithms are a special case of the algorithm \( \text{Move}_{gen} \) presented in Chapter 5, and they all have the following properties:

**Assumption 2.2.4**

1. At each component \( C \), there is a vertex \( v(C) \), s.t., after each block-move, only \( v(C) \) may contain blocks.

2. If \( \ell \geq 2 \), then the algorithm does not contain \( BM_{(a)} \).

**Definition 2.2.5** We define \( k(A, H) \) as the number of blocks used by block-algorithm \( A \) working for Hanoi graph \( H \).
Proposition 2.2.6 Any block-algorithm, \( A \), which has the properties of Assumption 2.2.4, runs by any Hanoi Graph \( H \) in AAG3 in time at most \( 2^{\pi(\frac{n}{k}) + c(\ell)} \), for some function \( c(\ell) \), where \( c(\ell) = O(\log^2 \ell) \).

Proof: Since \( H^{SCC} \) is a DAG, the number of times each block is moved along the components of the graph, is bounded by \( \ell - 1 \). Moreover, the total length of all move-paths, for each block is at most \( \ell - 1 \). We proved in Proposition 2.2.3 that the number of moves in a block-move for \( m \) disks is \((2^m - 1) + m \cdot (|p| + 1)\). Since we have \( k \) blocks, with at most \( \lceil \frac{n}{k} \rceil \) disks each, by Proposition 2.2.3 the number of steps is at most:

\[
\begin{align*}
k \cdot (\ell - 1) \cdot (2^{\lfloor \frac{n}{k} \rfloor} - 1) + k \cdot 2^{\ell} \cdot \lceil \frac{n}{k} \rceil & \leq k \ell \cdot 2^{\lfloor \frac{n}{k} \rfloor} + 3\ell n \\
& \leq 4 \cdot k \ell \cdot 2^{\lfloor \frac{n}{k} \rfloor} \\
& \leq 2^{\frac{n}{k} + c(\ell)},
\end{align*}
\]

where \( c(\ell) = O(\log^2 \ell) \). The second inequality in the above chain is equivalent to:

\[
\begin{align*}
3k\ell \cdot 2^{\lfloor \frac{n}{k} \rfloor} & \geq 3\ell n, \\
k \cdot 2^{\lfloor \frac{n}{k} \rfloor} & \geq n, \\
2^{\lfloor \frac{n}{k} \rfloor} & \geq \frac{n}{k}.
\end{align*}
\]

The last inequality follows from the known inequality \( 2^x > x \) for any \( x \geq 0 \).

The bound \( c(\ell) = O(\log^2 \ell) \) follows from the bound \( k(Move_{TC}, H^{TC}_\ell) = O(\ell^{1/2} \log \ell) \) proved in Theorem 3.2.1.

\( \Box \)

Observation 2.2.7 For any \( \Gamma' < \Gamma \), if there exists a block-algorithm moving \( \Gamma \) blocks from \( C_i \) to \( C_j \), then there exists a block-algorithm moving \( \Gamma' \) blocks from \( C_i \) to \( C_j \).
Indeed, the restriction of the given algorithm to the disks in the first $\Gamma'$ blocks is an algorithm as required.

## 2.3 Basic Results

In this section we study the basic graphs $H_1$, $H_2$ and $H_2^{\text{max}}$ (see Figure 2.2) with relation to block-moves. We would use the results in further chapters.

For graphs $H_2^{\text{max}}$ and $H_2$, we have chosen $S$ and $D$ to be $v_1$ and $u_1$ respectively, though the following claims could easily be adjusted to any source from $C_i$ and destination from $C_j$.

**Claim 2.3.1** For a given configuration, the maximal number of blocks, which a block-algorithm can transfer inside $H_1$ is one block.

**Proof:** Observation 2.2.2 implies that for $BM(a)$, we can transfer only the smallest block inside the component. Since only one such block exists, the correctness of the claim follows. □

**Claim 2.3.2** The maximal number of blocks, which a block-algorithm can transfer in $H_2$ and $H_2^{\text{max}}$ from $S$ to $D$ under Assumption 2.2.4, is one block.
Proof: A single block-move from $S$ to $D$, does not violate the assumption. For two blocks, the first block-move of $B_1$ must be to some vertex of $C_j$, at some point later $B_2$ needs to be moved to another vertex of $C_j$, a contradiction to Assumption 2.2.4.

Note that the following result is not restricted to block-moves.

Claim 2.3.3 The minimal number of moves required to move in $H_1$, $n$ disks from $v_1$ to some configuration in $v_2$ and $v_3$ is $2^{n-1}$.

Proof: When disk $n$ moves from $v_1$ to w.l.o.g $v_2$, all other disks are gathered on $v_3$. The minimal number of moves required to transfer a tower of size $n - 1$ from $v_1$ to $v_3$ is $2^{n-1} - 1$, we add to it the move of disk $n$, and we get $2^{n-1}$.

2.4 Auxiliary Mathematical Results

In this section we prove some basic mathematical results that would be useful in further chapters.

Lemma 2.4.1 Consider the following two recurrence relations $(a_n)_{n=1}^\infty$ and $(b_n)_{n=1}^\infty$ which satisfy:

$$a_n = a_{n-1} + ka_{\lceil \frac{n}{2} \rceil}$$

$$b_n = b_{n-1} + \lambda kb_{\lceil \frac{n}{2} \rceil}.$$  

For each $n \geq 1$, where $\lambda \geq 1$ and $k \geq 0$, $\frac{b_n}{a_n} \leq \frac{b_1}{a_1} \lambda^{[\log n]}$.

Proof: Let us define a new recurrence sequence, $(c_n)_{n=1}^\infty$, in the following way:

$$c_n = \frac{b_1}{a_1} \lambda^{[\log n]} a_n.$$
We claim that $c_n \geq b_n$ for each $n \geq 1$, hence

\[
\frac{b_n}{a_n} \leq \frac{c_n}{a_n} \leq \frac{b_1}{a_1} \lambda^{\log n}.
\]

The proof is by induction on $n$.

**Basis:** $n = 1$. Straightforward from definition.

**Induction Step:** Assume the correctness of the claim for all values of $n' < n$ and prove it for $n \geq 2$.

We distinguish between the following two cases:

1. $n \equiv 0 \pmod{2}$. From equation (2.1) we get

   \[
   c_n = \frac{b_1}{a_1} \lambda^{\log n} a_n = \frac{b_1}{a_1} \lambda^{\log n} a_{n-1} + \frac{b_1}{a_1} \lambda^{\log n} a_{\frac{n}{2}}
   \]

   \[
   = \frac{b_1}{a_1} \lambda^{\log n} a_{n-1} + \frac{b_1}{a_1} \lambda^{\log \frac{n}{2} + 1} a_{\frac{n}{2}}
   \]

   \[
   \geq c_{n-1} + \lambda k c_{\frac{n}{2}}.
   \]

2. $n \equiv 1 \pmod{2}$. In this case we get

   \[
   c_n = \frac{b_1}{a_1} \lambda^{\log n} a_n = \frac{b_1}{a_1} \lambda^{\log n} a_{n-1} + \frac{b_1}{a_1} \lambda^{\log n} a_{\frac{n+1}{2}}
   \]

   \[
   = \frac{b_1}{a_1} \lambda^{\log n} a_{n-1} + \frac{b_1}{a_1} \lambda^{\log \frac{n+1}{2} + 1} a_{\frac{n+1}{2}}
   \]

   \[
   \geq c_{n-1} + \lambda k c_{\frac{n}{2}}.
   \]

In either case we got that from the induction hypothesis

\[
\frac{b_n}{a_n} \leq \frac{c_n}{a_n} \leq \frac{b_1}{a_1} \lambda^{\log n}.
\]

\[
c_n \geq c_{n-1} + \lambda k c_{\frac{n}{2}} \geq b_{n-1} + \lambda k b_{\frac{n}{2}} = b_n.
\]

\[\square\]
Lemma 2.4.2 Let \( F(x_1, \ldots, x_k) = \sum_{i=1}^{k} f_i(x_i) \), where each \( f_i \) is a continuously differentiable convex function. The conditional minimum of \( F \) with the constraint \( \sum_{i=1}^{k} x_i = n \), is achieved at point \((x_1, x_2, \ldots, x_k)\) satisfying \( f'_1(x_1) = f'_2(x_2) = \ldots = f'_k(x_k) \).

Proof: Let us denote the constraint by \( g(x_1, \ldots, x_k) \), i.e.,
\[
g(x_1, \ldots, x_k) = \sum_{i=1}^{k} x_i - n = 0.
\]
Functions \( F \) and \( g \) have continuous first partial derivatives and \( \nabla g \neq 0 \) at any point (where \( \nabla g \) is the gradient). So, we can use Lagrange multipliers, to find the extremum of \( F \) subject to the constraint \( g \). So, there exists \( \lambda \), s.t.:
\[
\nabla F = -\lambda \nabla g,
\]
and we get the following \( k \) equations, \( \forall i, 1 \leq i \leq k \):
\[
\frac{\partial F}{\partial x_i} = \frac{df_i}{dx_i} = -\lambda.
\]
This shows that all \( f'_i \) are equal. From the fact that all \( f_i \) are convex functions, we infer that the extremum of \( F \) is a global minimum. \( \square \)

We note that a discrete minimum for \( F \) is not necessarily equal to the real minimum rounded with ceilings and floors. For example, for the function \( F = \sum_{i=1}^{6} 5x_i^2 + x_7^2 \) with \( n = 35 \), the minimum is achieved at \( x_i = 3 \frac{2}{11} \) for \( 1 \leq i \leq 6 \), and \( x_7 = 15 \frac{10}{11} \), whereas the discrete minimum is achieved at \( x_i = 3 \) for \( 1 \leq i \leq 6 \), and \( x_7 = 17 \).
Consider a function \( F(n_1, \ldots, n_k) = \sum_{i=1}^{k} f_i(n_i) \), where each \( f_i \) satisfy \( f_i(n_i) \geq 2^{\frac{n_i}{s_1} + b_i} \), \( \sum_{i=1}^{k} n_i = n \), and \( a_i \) and \( b_i \) are some constants.

1. The minimal value of \( F \), \( \min F \), satisfies \( \min F \geq 2^{\frac{n}{s_1} - \frac{a_2}{s_1} \log s_1} \), where \( s_1 = \sum_{i=1}^{k} a_i \) and \( s_2 = \sum_{i=1}^{k} a_i (\log a_i - b_i) \).

2. In the special case where \( \forall i, a_i = a, b_i = b \), \( \min F \geq 2^{\frac{n}{s_1} + b} \).

**Proof:** Let \( g_i(n_i) = 2^{\frac{a_i}{s_1} + b_i} \), for each \( 1 \leq i \leq k \), and \( G(n_1, \ldots, n_k) = \sum_{i=1}^{k} g_i(n_i) \). From Lemma 2.4.2, we get that any vector \((n_1, n_2, \ldots, n_k)\) which yields the minimal value for \( G \), satisfies \( g_i'(n_1) = g_2'(n_2) = \ldots = g_k'(n_k) \).

Let \( s_1 = \sum_{i=1}^{k} a_i \) and \( s_2 = \sum_{i=1}^{k} a_i (\log a_i - b_i) \).

We set
\[
n_i = \frac{a_i(n - s_2)}{s_1} + a_i (\log a_i - b_i).
\]

The vector \((n_1, n_2, \ldots, n_k)\) has the following properties.

1. \[ \sum_{i=1}^{k} n_i = \sum \left( \frac{a_i (n - s_2)}{s_1} + a_i (\log a_i - b_i) \right) = \sum \frac{a_i (n - s_2)}{s_1} + s_2 = n. \]

2. \( \forall i, g_i'(n_i) = \frac{\ln 2}{a_i} \cdot 2^{\frac{a_i}{s_1} + b_i} = \frac{\ln 2}{a_i} \cdot 2^{\frac{a_i}{s_1} + (\log a_i - b_i) + b_i} = \ln 2 \cdot 2^{\frac{a_i}{s_1}}. \) Thus,
\[ g_1'(n_1) = g_2'(n_2) = \ldots = g_k'(n_k). \]

According to Lemma 2.4.2, our choice of \( n_i \) provides the minimum of \( G \).

Substituting it in \( G \) we get:
\[
\min F \geq \min G \geq \sum_{i=1}^{k} \left( 2^{\left( \frac{a_i (n - s_2)}{s_1} + a_i (\log a_i - b_i) \right) \frac{1}{s_1} + b_i} \right) = \sum_{i=1}^{k} \left( 2^{\frac{n}{s_1} + \left( \log a_i - \frac{a_2}{s_1} \right)} \right)
\]
\[
= \left( \sum_{i=1}^{k} 2^{\log a_i} \right) \cdot 2^{\frac{n}{s_1} - \frac{a_2}{s_1}} = s_1 \cdot 2^{\frac{n}{s_1} - \frac{a_2}{s_1}} = 2^{\frac{n}{s_1} - \frac{a_2}{s_1} + \log s_1}.
\]
In the case of item 2 we get:

\[ 2^n \frac{n}{s_1} - \frac{n}{s_1} \log s_1 = 2^n \frac{n}{a} - \frac{k \cdot a \log (a-b)}{k} + \log k \cdot a = 2^n \frac{n}{b} + b + \log k \geq 2^n \frac{n}{b} + b. \]

\[ \square \]

As a straightforward consequence, also the minimum value of \( F \) over the integer vectors satisfies the same inequalities.
Chapter 3

Bounds for Maximal Almost-Acyclic Hanoi Graphs

Chapter 3 has the following structure:

1. In Section 3.1, we find the Hanoi graph $H_{TC}^\ell$ with $\ell$ components, s.t. the number of steps for it is minimal among all graphs in $AAG3_\ell$.

2. In Section 3.2, we suggest an algorithm solving $H_{TC}^\ell$ and develop a recurrence for the number of its steps, which is an upper bound for the best solution of $H_{TC}^\ell$.

3. In Section 3.3, we define a recurrence for a lower bound on the number of steps required for solving $H_{TC}^\ell$, and thus of any Hanoi graph in $AAG3_\ell$.

4. In Section 3.4, we show that the solutions to the recurrences developed above for our upper and lower bounds (which remain implicit, as for now) form a tight pair, by bounding their ratio.
5. In Section 3.5, we find explicit upper and lower bounds for the number of steps required for solving $H^{TC}_\ell$, based on the above recurrences.

## 3.1 Best Almost-Acyclic Hanoi Graph $H^{TC}_\ell$

Let us define the transitive chain graph $H^{TC}_\ell$:

$$\begin{align*}
V(H^{TC}_\ell) &= \{v_{ij} : 1 \leq i \leq \ell, 1 \leq j \leq 3\} \\
E(H^{TC}_\ell) &= \{(v_{ij}, v_{st}) : (i \leq s)\}.
\end{align*}$$

The number of SCCs in $H^{TC}_\ell$ is $\ell$, where SCC $C_i = \{v_{i1}, v_{i2}, v_{i3}\}$. In Figure 3.1 we depict $H^{TC}_4$. To avoid over-congestion, we draw only the edges between vertices belonging to the same $C_i$ or to consecutive $C_i's$, but there are actually also edges from each vertex to each vertex below it.

**Proposition 3.1.1** The running time required to solve $HAN(H^{TC}_\ell, n)$ is minimal over all almost-acyclic Hanoi graphs with $\ell$ SCCs.
**Proof:** Notice that adding edges or vertices to Hanoi graph $H$ can only decrease the number of steps required to solve $HAN(H,n)$. For an arbitrary Hanoi graph with $\ell(H) = \ell$, we can perform a topological sort $V^{SCC} \rightarrow [1..\ell]$ of $H^{SCC}$. Let us denote the SSC’s by $C_i$, $i \in [1..\ell]$, accordingly. We denote the vertices in $C_i(H)$ by $v_{ik}$, for $1 \leq k \leq |V(C_i)|$. Since $H \in AAG3_\ell$, $|V(C_i)| \leq 3$ for any $1 \leq i \leq 3$. By the definition of $H^{TC}_\ell$, there is a straightforward injection from $H$ to $H^{TC}_\ell$, i.e., $H$ is isomorphic to a subgraph of $H^{TC}_\ell$, thus the lemma follows. 

3.2 Algorithm for $H^{TC}_\ell$ and Its Number of Steps

In this section we show a block-algorithm solving $HAN(H^{TC}_\ell)$, and analyze it in Theorem 3.2.1. We denote the number of blocks that Algorithm 3.1 transfers in $H^{TC}_\ell$ from $S$ to $D$ by $b_\ell$. Using the notation of Definition 2.1.1, we define $H^{TC}_{\ell,ij} = H^{TC}_\ell[C_i,C_j]$:

$$V(H^{TC}_{\ell,ij}) = \{v_{st} : i \leq s \leq j, 1 \leq t \leq 3\},$$

$$E(H^{TC}_{\ell,ij}) = \{(v_{st},v_{xy}) : (s \leq x)\}.$$ 

Note that $H^{TC}_{\ell,1\ell} = H^{TC}_\ell$, and that $H^{TC}_{\ell,ij}$ is isomorphic to $H^{TC}_{j-i+1}$. Notice that $H^{TC}_\ell \setminus \{C_i\}$ is isomorphic to $H^{TC}_{\ell-1}$.

**Theorem 3.2.1** 1. Algorithm 3.1 solves $HAN(H^{TC}_\ell,n)$, while satisfying having the properties of Assumption 2.2.4 for any $\ell$ and $n$. 

2. The number of blocks used by Algorithm 3.1 is defined by the recurrence:

\[
\begin{align*}
b_2 &= b_1 = 1 \\
b_\ell &= b_{\ell-1} + b_{\lceil \frac{\ell}{2} \rceil}, \quad \ell \geq 3,
\end{align*}
\]

so that the number of its steps is at most \(2^{n^* + c(\ell)}\), for some function \(c(\ell)\).

3. For each \(\varepsilon > 0\) there exist a constant \(C_\varepsilon > 0\), such that \(b_\ell \geq C_\varepsilon \ell^{(\frac{1}{2} - \varepsilon) \log \ell}\) for each \(\ell \geq 1\).

**Algorithm 3.1** \(\text{Move}_{tc}(H^\text{TC}_{1, \ell}, S, D)\)

1: if \(\ell = 1\) or \(\ell = 2\) then
2: transfer one block from \(S\) to \(D\) using \(BM(a)\) or \(BM(b)\) accordingly.
3: else
4: \(\text{Move}_{tc}(H^\text{TC}_{1, \lceil \frac{\ell}{2} \rceil}, S, v_{\lceil \frac{\ell}{2} \rceil}^1)\).
5: \(\text{Move}_{tc}(H^\text{TC}_{\ell, \ell - 1 \{C_{\lceil \frac{\ell}{2} \rceil}\}}, S, D)\).
6: \(\text{Move}_{tc}(H^\text{TC}_{\ell, v_{\lceil \frac{\ell}{2} \rceil}^1}, D)\).
7: end if

**Proof:** First we prove item 1. The proof is by induction on \(\ell\). For \(\ell = 1\) and \(\ell = 2\) the statement is correct according to Observation 2.2.2. Let’s assume its correctness for all \(\ell' < \ell\) and prove correctness for \(\ell\). In Line 4 we transfer \(b_{\lceil \frac{\ell}{2} \rceil}\) blocks from \(S\) to \(v_{\lceil \frac{\ell}{2} \rceil}^1\). In Line 5, we transfer in the graph \(H^\text{TC}_{\ell, \ell - 1 \{C_{\lceil \frac{\ell}{2} \rceil}\}}\), which is a transitive-chain graph of size \(\ell - 1\), \(b_{\ell - 1}\) blocks from \(S\) to \(D\). Finally, we transfer the \(b_{\lceil \frac{\ell}{2} \rceil}\) blocks residing on \(v_{\lceil \frac{\ell}{2} \rceil}^1\) to \(D\). All the above transfers are legal due to the induction hypothesis. It is easy to see that the algorithm has the properties of Assumption 2.2.4. Thus, item 1 follows.
By the above proof, we get the recursion, as in item 2. From Proposition 2.2.6, we know that the running time of Move_{tc}(H_{TC}^\ell, S, D) is $2^{\frac{n}{b\ell} - c(\ell)}$, for some function $c(\ell)$. Correctness of item 3 is proven in Section 3.5.1. □

By Theorem 3.2.1, $O_\ell(2^{\frac{n}{b\ell}})$ is an upper bound for solving $H_{TC}^\ell$.

### 3.3 Lower Bound for a General Algorithm

This section develops a lower bound for the number of steps needed to solve $H_{TC}^\ell$, and thus any Hanoi graph in $AAG_{3\ell}$.

**Theorem 3.3.1** 1. The minimal number of steps required to solve $HAN(H_{TC}^\ell, n)$ by any algorithm is $2^{\frac{n}{b\ell} - c(\ell)}$, for some function $c(\ell)$, where $(a_\ell)_{\ell=1}^\infty$ is defined by the recurrence:

\[
\begin{align*}
a_1 &= 1, a_2 = 2, \\
a_\ell &= a_{\ell-1} + 3a_{\lceil \frac{\ell}{2} \rceil}, \quad \ell \geq 3.
\end{align*}
\]

2. There exist a constant $C$, such that $a_\ell \leq C\ell^{\frac{1}{3}\log \ell}$, for each $\ell \geq 1$.

**Proof:**

**Lemma 3.3.2** If in an empty $H_{TC}^\ell$, at least $2^{\frac{n}{b\ell} - c(\ell)}$ steps are required to transfer $n$ disks from $v_{ik_1}$ to $v_{jk_2}$, where $k_1, k_2 \in \{1, 2, 3\}$, then at least $2^{\frac{n}{b\ell} - c(\ell)}$ steps are required to transfer $n$ disks from $v_{ik_1}$ to some configuration in $C_j$, or to transfer $n$ disks from any configuration in $C_i$ to $v_{jk_2}$.

**Proof:** Let Alg be some sequence transferring $n$ disks from $v_{ik_1}$ to $C_j$. Let us look at the configuration just after Alg finished executing, and denote the set of disks at $v_{jh}$ by $A_h$, $|A_h| = n_h$, for $h \in \{1, 2, 3\}$. 


By the assumption of our lemma and Proposition 2.4.3(2), \( |\text{Alg}| = \sum_{h=1}^{3} |(\text{Alg}|_{A_h})| \geq \sum_{h=1}^{3} 2^{\frac{n}{3}c_2(\ell)} \geq 2^{\frac{n}{3}c_2(\ell)}.

The second argument follows from symmetry. \(\square\)

**Fact 3.3.3** The recurrence \((a_\ell)_{\ell=1}^\infty\) is equivalently defined by \(a_1 = 1, a_2 = 2,\) and for \(\ell \geq 3:\)

\[
a_\ell = \begin{cases} 
6 \sum_{i=2}^{\lfloor \frac{\ell}{2} \rfloor} a_i + 2 & \ell \equiv 0 \pmod{2}, \\
6 \sum_{i=2}^{\lfloor \frac{\ell}{2} \rfloor} a_i + 3a_{\lfloor \frac{\ell}{2} \rfloor} + 2 & \ell \equiv 1 \pmod{2}.
\end{cases}
\]  

(3.1)

**Proof:** For an even \(\ell,\)

\[
a_{\ell-1} + 3a_{\lfloor \frac{\ell}{2} \rfloor} = 6 \sum_{i=2}^{\lfloor \frac{\ell-1}{2} \rfloor} a_i + 3a_{\lfloor \frac{\ell-1}{2} \rfloor} + 3a_{\lfloor \frac{\ell}{2} \rfloor} + 2 = 6 \sum_{i=2}^{\lfloor \frac{\ell}{2} \rfloor} a_i + 2 = a_\ell.
\]

and for an odd \(\ell,\)

\[
a_{\ell-1} + 3a_{\lfloor \frac{\ell}{2} \rfloor} = 6 \sum_{i=2}^{\lfloor \frac{\ell-1}{2} \rfloor} a_i + 3a_{\lfloor \frac{\ell}{2} \rfloor} + 2 = a_\ell.
\]

\(\square\)

**Proposition 3.3.4** The number of moves required to solve \(HAN(H^T_{TC}, n)\) is at least \(2^{\frac{n}{3}c_1(\ell)}\), for some function \(c(\ell)\), where \(a_\ell\) is defined by Recurrence (3.1).

**Proof:** The proof is by induction on \(\ell\).

**Basis:** \(\ell = 1, 2.\)

For \(\ell = 1\) we have the classic problem \(\text{Recursive3Peg}\), requiring \(2^n - 1\) steps to solve.
For $\ell = 2$, let us look at the configuration of $D_n \setminus \{n\}$ during the first move of disk $n$. Let us denote the number of disks in $C_1$ by $q$, and thereby the number of disks in $C_2$ is $n - q - 1$.

From Claim 2.3.3, the minimal number of moves required to move the $q$ disks in $C_1$ before the first move of disk $n$, is at least $2^{q-1}$. In a similar way the minimal number of moves required to move the $n - q - 1$ disks in $C_2$, after the last move of disk $n$ is $2^{n-q-2}$. Altogether we have $2^{q-1} + 2^{n-q-2}$ moves. A simple calculation shows that the minimal value of this function is $2^{\lceil \frac{n+1}{2} \rceil} + 2^{\lfloor \frac{n+1}{2} \rfloor} \leq 2^\frac{n}{2}$.

**Induction Step:** Assume the correctness of the claim for all values of $\ell' < \ell$ and for all $n$, and prove it for $\ell$ and for all $n$.

Let us look at the configuration of $D_n \setminus \{n\}$ during the first move of disk $n$. Let us denote by $n_i$, the number of disks in $C_i$ for each $i \in \{1, \ldots, \ell\}$. By Claim 2.3.3, the minimal number of moves required to transfer $n_1$ disks from $v_{11} = S$ to some configuration in $C_1$ is $2^{n_1-1}$. From symmetry, the minimal number of moves required to transfer $n_\ell$ disks from some configuration in $C_\ell$, to $v_{\ell l} = D$ is $2^{n_\ell-1}$.

From the induction hypothesis, the number of steps required to solve $HAN(H^{TC}_{11}, n_i)$, for $2 \leq i \leq \lceil \frac{\ell}{2} \rceil$, is at least $2^{\frac{n_1}{\ell} - c(1)}$, for some function. Applying Lemma 3.3.2, we get that number of moves required to move $n_i$ disks from $S$ to $C_i$ is $2^{\frac{n_1}{\ell} - c(1)}$. From symmetry, the number of moves required to transfer $n_i$ disks from $C_i$ to $D$, for $\lceil \frac{\ell}{2} \rceil < i \leq \ell - 1$, is $2^{\frac{n_1}{\ell+i-1} - c(\ell+1-i)}$. 


Let $s(\ell) = \sum_{i=2}^{\lceil \frac{\ell}{2} \rceil} 3a_i \left( \log a_i + c(i) \right) + \sum_{i=\lceil \frac{\ell}{2} \rceil + 1}^{\ell - 1} 3a_{i+1-i} \left( \log a_{i+1-i} + c(\ell + 1 - i) \right) - 2$.

Altogether, we get from Proposition 2.4.3 (1) the following lower bound:

$$\sum_{i=2}^{\lceil \frac{\ell}{2} \rceil} 2^{\frac{n}{a_\ell} - c(i)} + \sum_{i=\lceil \frac{\ell}{2} \rceil + 1}^{\ell - 1} 2^{\frac{n}{a_{i+1-i}} - c(\ell + 1 - i)} + 2^{n_1-1} + 2^{n_\ell-1} \geq 2^{\frac{n}{a_\ell} - \frac{s(\ell)}{a_\ell} + \log a_\ell} = 2^{\frac{n}{a_\ell} - \frac{s(\ell) + \log a_\ell}{a_\ell}} = 2^{\frac{n}{a_\ell} - c(\ell)}.$$ 

where $c(\ell) = \frac{s(\ell) + 1}{a_\ell} + \log a_\ell$ and $a_\ell$ satisfies:

$$a_\ell = \begin{cases} 
2 \sum_{i=2}^{\lceil \frac{\ell}{2} \rceil} 3a_i + 2 & \ell \equiv 0 \pmod{2}, \\
2 \sum_{i=2}^{\lceil \frac{\ell}{2} \rceil} 3a_i + 3a_{\lceil \frac{\ell}{2} \rceil} + 2 & \ell \equiv 1 \pmod{2}.
\end{cases}$$

Correctness of item 1 of Theorem 3.3.1 follows from Proposition 3.3.4, and Fact 3.3.3. Correctness of item 2 is proven in Section 3.5.2.

3.4 Measuring the Tightness of Our Bounds

Given $H_\ell^{TC}$, we get from Proposition 3.3.4 a lower bound $\Omega_\ell(2^{\frac{n}{a_\ell}})$ on the number of moves required to solve $HAN(H, n)$. On the other hand, we introduced a block-algorithm solving the problem with $O_\ell(2^{\frac{n}{b_\ell}})$ steps. We show in this section that $(a_\ell)_{\ell=2}^\infty$ and $(b_\ell)_{\ell=2}^\infty$, are in fact very close to each other.
Claim 3.4.1  For each $\ell \geq 2$, \( \frac{a_\ell}{b_\ell} \leq 2 \cdot 3^{\lceil \log \ell \rceil} < 6 \cdot \ell^{\log 3} \).

Proof: From Theorem 3.2.1 and Theorem 3.3.1 (1) we get:

\[
\begin{align*}
a_2 &= 2, b_2 = 1, \\
b_\ell &= b_{\ell-1} + b_{\lfloor \frac{\ell}{2} \rfloor}, \quad \ell \geq 3, \\
a_\ell &= a_{\ell-1} + 3a_{\lfloor \frac{\ell}{2} \rfloor}, \quad \ell \geq 3.
\end{align*}
\]

A simple application of Lemma 2.4.1 with $k = 1$ and $\lambda = 3$ yields the claim.  

We note that $6 \cdot \ell^{\log 3}$ is a low degree polynomial of $\ell$, which is a term of a lesser order than the order of the ratio of our previous upper and lower bounds $\frac{C_\varepsilon \ell^{\frac{1}{2} \log \ell}}{C_\varepsilon \ell^{(\frac{1}{2} - \varepsilon) \log \ell}} = \frac{C_\varepsilon}{C_\varepsilon} \cdot \ell^{\varepsilon \log \ell}$.

3.5 Proofs of Theorems 3.2.1(2) and 3.3.1(2)

The techniques used in this section to acquire the lower and upper bounds are based on those in D. Azriel’s work on Unsolvable Hanoi Graphs [5].

3.5.1 Proof of Theorem 3.2.1(2)

Proof: Let $\varepsilon > 0$. Obviously, the lemma is true for $\varepsilon \geq \frac{1}{2}$, so we may assume $\varepsilon < \frac{1}{2}$. Set

\[ \alpha = \frac{1 - 2\varepsilon}{2 \ln 2} \]

and let $L_0$ be the minimal number $\ell$ for which $\frac{\ell^\varepsilon}{\ln(\ell+1)} \geq \frac{C_\varepsilon}{2^{\alpha \ln 2}}$, where $C$ is a constant to be determined. Take $C_\varepsilon$ so that the required inequality holds.
for all $\ell < L_0$. We proceed by induction on $\ell$. Assume that $b_k \geq C_\varepsilon k^{\alpha \ln k} = C_\varepsilon k^{(\frac{1}{2} - \varepsilon) \log k}$ for each $0 \leq k \leq \ell$, for some $\ell \geq L_0$. By Theorem 3.2.1 (2) and the induction hypothesis:

$$b_{\ell+1} \geq b_\ell + b_{\left\lfloor \frac{\ell+1}{2} \right\rfloor} \geq C_\varepsilon e^{\alpha \ln \ell} + C_\varepsilon \left(\frac{\ell + 1}{2}\right)^{\alpha \ln \left(\frac{\ell+1}{2}\right)}.$$ 

Thus it suffices to prove that:

$$\ell^{\alpha \ln \ell} + \left(\frac{\ell + 1}{2}\right)^{\alpha \ln \left(\frac{\ell+1}{2}\right)} \geq (\ell + 1)^{\alpha \ln (\ell + 1)}, \quad \ell \geq L_0. \tag{3.2}$$

The second term on the left-hand side of (3.2) may be bounded from below by:

$$\left(\frac{\ell + 1}{2}\right)^{\alpha \ln \left(\frac{\ell+1}{2}\right)} \geq \left(\frac{\ell}{2}\right)^{\alpha \ln \left(\frac{1}{2}\right)} = \frac{\ell^{\alpha \ln \ell} 2^{\alpha \ln 2}}{\ell^{2\alpha \ln 2}} \tag{3.3}$$

Using the inequality $\ln(1 + 1/\ell) \leq 1/\ell$, we estimate the right-hand side of (3.2),

$$\ell^{\alpha \ln (\ell + 1)} = \ell^{\alpha \ln \ell} e^{\alpha \ln (1 + \frac{1}{\ell})} \ln (\ell + 1) \leq \ell^{\alpha \ln \ell} e^{\alpha \ln (\ell + 1)} \leq \ell^{\alpha \ln \ell} e^{\alpha \ln (\ell + 1)} = e^{\alpha \ln \ell} \left[1 + O\left(\frac{\ln (\ell + 1)}{\ell}\right)\right] \tag{3.4}$$
for a suitable $C$. To prove (3.2) it suffices, by (3.3) and (3.4), to show that the following inequality holds:

$$\ell^{\alpha \ln\ell} \left[ 1 + 2^{\alpha \ln^2 1} \frac{1}{\ell^{2\alpha \ln^2}} \right] \geq \ell^{\alpha \ln\ell} \left[ 1 + C \frac{\ln(\ell + 1)}{\ell} \right], \quad \ell \geq L_0.$$ 

It is equivalent to the inequality

$$\frac{\ell^{1-2\alpha \ln^2}}{\ln(\ell + 1)} = \frac{\ell^{2\alpha}}{\ln(\ell + 1)} \geq \frac{C}{2^{\alpha \ln^2}}, \quad \ell \geq L_0,$$

which is indeed correct, due to the way we chose $L_0$. 

\[\square\]

### 3.5.2 Proof of Theorem 3.3.1(2)

**Proof:** We would prove by induction that $a_\ell \leq C\ell^{\frac{1}{2}} \log^\ell$, for each $\ell \geq 1$.

Set

$$\alpha = \frac{1}{2 \ln 2}$$

and let $L_0$ be the minimal number $\ell$ for which the constant $\frac{3\sqrt{2} e^{4\alpha t_0} (\frac{1}{2} + 1)}{\alpha} \leq \ln \ell(\ell + 1)$. Take $C$ so that the required inequality holds for all $\ell < L_0$.

Assume inductively that $a_k \leq Ck^{\alpha \ln k} = Ck^{\frac{1}{2} \log k}$ for each $0 \leq k \leq \ell$ for some $\ell \geq L_0$. By the induction hypothesis and Theorem 3.3.1(2):

$$a_{\ell+1} = a_\ell + 3a_{\left\lceil \frac{\ell+1}{2} \right\rceil} \leq C\ell^{\alpha \ln \ell} + 3C\Bigg[ \frac{\ell + 1}{2} \Bigg]^{\alpha \ln\left[\frac{\ell+1}{2}\right]} \quad \text{(3.5)}$$

$$\leq C\ell^{\alpha \ln \ell} + 3C \left( \frac{\ell + 2}{2} \right)^{\alpha \ln\left[\frac{\ell+2}{2}\right]}.$$
Thus it suffices to prove that:

\[ C\ell^\alpha + 3C \left( \frac{\ell + 2}{2} \right)^{\alpha \ln\left( \frac{\ell + 2}{2} \right)} \leq C(\ell + 1)^{\alpha \ln(\ell + 1)}, \quad \ell \geq L_0. \quad (3.6) \]

The right-hand side may be bounded from below using Bernoulli inequality:

\[(1 + x)^a > 1 + ax \text{ for each } x > -1 \neq 0 \text{ and } a > 1.\]

and the simple equality \( a^{\ln b} = b^{\ln a} \).

\[(\ell + 1)^{\alpha \ln(\ell + 1)} = \ell^{\alpha \ln\ell} \left(\frac{\ln(\ell + 1)}{\ln\ell}\right)^{\alpha \ln(\ell + 1)} = \ell^{\alpha \ln\ell} \ell^{\alpha \ln\left(\frac{\ln(\ell + 1)}{\ln\ell}\right)}^{\alpha \ln(\ell + 1)} = \ell^{\alpha \ln\ell} \left(1 + \frac{\alpha \ln(\ell + 1)}{\ell}\right) \quad (3.7)\]

We bound from above a part of the left-hand side of equation (3.6)

\[ \left( \frac{\ell + 2}{2} \right)^{\alpha \ln\left( \frac{\ell + 2}{2} \right)} = \left( \frac{\ell + 2}{2} \right)^{\alpha \ln(\ell + 2)} \frac{2^{\alpha \ln 2}}{2^{\alpha \ln(\ell + 2)}} = \left( \frac{\ell + 2}{2} \right)^{\alpha \ln(\ell + 2)} \sqrt{2} \leq \left( \frac{\ell + 2}{\ell + 2} \right)^{\alpha \ln(\ell + 2)} \sqrt{2} \leq \frac{\ell + 2}{\ell + 2} \left( \frac{\ell + 2}{\ell + 2} \right)^{\alpha \ln(\ell + 2)} \sqrt{2}, \]

and we have

\[ (\ell + 2)^{\alpha \ln(\ell + 2)} = \ell^{\alpha \ln\ell} \left(1 + \frac{2}{\ell}\right)^{\alpha \ln(\ell + 2)} \ell^{\alpha \ln\left(\frac{\ln(\ell + 2)}{\ln\ell}\right)} = \ell^{\alpha \ln\ell} e^{F(\ell)}, \]

where

\[ F(\ell) = \alpha \ln \left(1 + \frac{2}{\ell}\right) \ln \left[ \ell \left(1 + \frac{2}{\ell}\right) \right] + \alpha \ln \ell \ln \left(1 + \frac{2}{\ell}\right) = \alpha \ln \left(1 + \frac{2}{\ell}\right) \left[ 2\ln \ell + \ln \left(1 + \frac{2}{\ell}\right) \right]. \]
We bound $F(\ell)$ by using the inequalities $\ln(1 + \frac{2}{\ell}) \leq \frac{2}{\ell}$ and $0 \leq \frac{\ln \ell}{\ell} \leq 1/e$ for $\ell \geq 1$:

$$F(\ell) \leq \frac{4\alpha}{\ell} \left[ \ln \ell + \frac{1}{\ell} \right] \leq 4\alpha \left[ \frac{1}{e} + \frac{1}{\ell^2} \right] \leq 4\alpha \left[ \frac{1}{e} + 1 \right].$$

Altogether we have:

$$3 \left( \frac{\ell + 2}{2} \right)^{\alpha \ln \left( \frac{\ell + 2}{2} \right)} \leq 3 \frac{\ell^{\alpha \ln \ell} e^{4\alpha (\frac{1}{\ell} + 1) \sqrt{2}}}{\ell} = C_0 \frac{\ell^{\alpha \ln \ell}}{\ell}. \quad (3.8)$$

Replace the right-hand side of (3.6) by that of (3.7) and the second term on the left-hand side by the right-hand side of (3.8). We get the stronger inequality:

$$\ell^{\alpha \ln \ell} + C_0 \frac{\ell^{\alpha \ln \ell}}{\ell} \leq \ell^{\alpha \ln \ell} \left( 1 + \frac{\alpha \ln \ell (\ell + 1)}{\ell} \right), \quad \ell \geq L_0,$$

which holds since it is equivalent to:

$$\frac{C_0}{\alpha} = \frac{3\sqrt{2} e^{4\alpha (\frac{1}{\ell} + 1)}}{\alpha} \leq \ln \ell (\ell + 1), \quad \ell \geq L_0.$$
Chapter 4

Complexity of Parallel-Components Hanoi Graphs

Denote by $PC_\ell$ the family of Hanoi graphs with exactly $\ell \geq 1$ disjoint complete graphs $\{C_1, C_2, \ldots, C_\ell\}$, of size three each, two vertices $S, D$, a directed edge from $S$ to each component, and another directed edge adjacent to the former edge, from each component to $D$, as depicted in Figure 4.1. Our goal is to show an algorithm solving $PC_\ell$ and to prove it’s optimality. We generalize this work in Section 4.2.

4.1 Exact bounds for $PC_\ell$

For the sake of completeness we give the classic algorithm, solving the original problem presented in the introduction.

Let us examine the Hanoi graph $H_1$ depicted in Figure 4.1. It is proven in [7] that the following $Move_{H_1}$ is an optimal algorithm to transfer $n$ disks
Figure 4.1: Hanoi Graphs $PC_\ell$, $PC_1$ and other graphs.

Algorithm 4.1 $Recursive3peg(n, S, D, A)$

1: if $n \geq 1$ then
2:    $Recursive3peg(n - 1, S, A, D)$.
3:    $S \xrightarrow{n} D$
4:    $Recursive3peg(n - 1, A, D, S)$.
5: end if
from $S$ to $D$ (due to symmetry also to $P_2$), and it requires $2^n + n - 1$ steps.

Algorithm 4.2 Move$_{H_1}(n, S, D)$

1: if $n > 0$ then
2: Move$_{H_1}(n - 1, S, P_2)$.
3: $S \xrightarrow{n} P_1; \ P_1 \xrightarrow{n} D$.
4: Recursive3peg($n - 1, P_2, D, P_1$).
5: end if

Now let's look at $H_1'$, as depicted in Figure 4.1, with the complete component denoted as $C_1$. This time our goal is to move the disks from a perfect configuration on peg $S$ to "some" legal configuration in which all disks reside in $C_1$ (i.e. pegs $P_1$, $P_2$ and $P_3$).

Proposition 4.1.1 The following algorithm Move$_{H_1'}(n, S, C_1)$ is optimal, and it takes $2^{n-2} + n$ moves, for $n \geq 2$.

Algorithm 4.3 Move$_{H_1'}(n, S, C_1)$

1: if $n = 1$ then
2: $S \xrightarrow{n} P_2$
3: else if $n = 2$ then
4: $S \xrightarrow{n-1} P_2; \ P_2 \xrightarrow{n-1} P_3$.
5: $S \xrightarrow{n} P_2$.
6: else
7: Move$_{H_1}(n - 2, S, P_3)$.
8: $S \xrightarrow{n-1} P_2; \ P_2 \xrightarrow{n-1} P_1$.
9: $S \xrightarrow{n} P_2$.
10: end if

Proof: Let Alg$_1(n)$ be some optimal algorithm, transferring $n$ disks from a perfect configuration on peg $S$ to "some" legal configuration in which all
disks reside in $C_1$. $Alg_1(n)$ must contain the move $S \overset{n}{\rightarrow} P_2$, and since the algorithm is optimal, every move after this move is redundant. So this is the last move of $Alg_1(n)$. During the move, peg $P_2$ must be free.

So $Alg_1$ is composed of an optimal move of $n - 1$ disks from $S$ to $P_1$ and $P_3$, and the single move $S \overset{n}{\rightarrow} P_2$. Let’s denote the former by $Alg_2(n - 1)$.

In $Alg_2$, we must have the following two moves:

1. $S \overset{n-1}{\rightarrow} P_2$
2. $A_2 \overset{n-1}{\rightarrow} P_i$ (i=1 or 3, wlog i=1).

During the second move, all other disks must be gathered on peg $P_3$. So we have that $|Alg_2(n-1)| \geq |Move_{H_1}(n-2, S, P_3)| + 2$. Therefore $|Alg_1(n)| \geq |Alg_2(n-1)| + 1 \geq |Move_{H_1}(n-2, S, P_3)| + 2 + 1 = 2^{n-2} + (n - 2) - 1 + 3 = 2^{n-2} + n$.

We have proven that each optimal algorithm solving the problem, contains at least $2^{n-2} + n$ moves. Since $|Move_{H_1}| = 2^{n-2} + n$, $Move_{H_1}$ is optimal.

Now let us examine the $PC_1$ case, as depicted in Figure 4.1.

**Proposition 4.1.2** The algorithm $Move_{PC_1}(n, S, D, C_1)$, which is a composition of $Move_{H_1}(n, S, C_1)$ and $Move_{H_1^*}(n, D, C_1)$ is an optimal algorithm for $PC_1$, and it takes $2^{n-1} + 2n$ moves, for $n \geq 2$.

**Proof:** Let’s prove that the proposed algorithm is indeed optimal. Let $Alg$ be some algorithm, and let us divide $Alg$ into 3 parts:
1. \( \text{Alg}[S \xrightarrow{1} P_2, S \xrightarrow{n} P_2] \).

2. \( \text{Alg}(S \xrightarrow{n} P_2, P_2 \xrightarrow{n} D) \).

3. \( \text{Alg}[P_2 \xrightarrow{n} D, P_2 \xrightarrow{1} D] \).

We notice the facts, that right after the first part, and right before the third part of the algorithm, all disks reside in \( C_1 \). Thus, due to Proposition 4.1.1:

1. \(|\text{Alg}[S \xrightarrow{1} P_2, S \xrightarrow{n} P_2]| \geq |\text{Move}_{H_1}(n, S, C_1)|\).

2. \(|\text{Alg}[P_2 \xrightarrow{n} D, P_2 \xrightarrow{1} D]| \geq |\text{Move}_{H_1}(n, D, C_1)|\).

This implies \(|\text{Alg}| \geq |\text{Move}_{PC_1}(n, S, D, C_1)|\), as required. This also implies that the only difference of an arbitrary optimal algorithm for \( PC_1 \) from \( \text{Move}_{PC_1} \) could be in choosing another optimal algorithms, instead of \( \text{Move}_{H_1} \) and \( \text{Move}_{H_1}' \), and nothing between them.

\(|\text{Move}_{PC_1}| = |\text{Move}_{H_1}| + |\text{Move}_{H_1}'| \) by definition. \(|\text{Move}_{H_1}| = |\text{Move}_{H_1}'| = (2^{n-2} + n) \) by Proposition 4.1.1. Therefore \(|\text{Move}_{PC_1}| = 2 \times (2^{n-2} + n) = 2^{n-1} + 2n\).

\(\square\)

Let \( \text{Alg} \) be some algorithm solving \( PC_\ell \), and let \( A_i \) be the group of disks that resided at some time during the algorithm in \( C_i \). Let \( n_i = |A_i| \), and \( f(A_i) = |(\text{Alg}|_{A_i})| \). Then we have the following facts.
1. \( \forall i \neq j, A_i \cap A_j = \emptyset \). This stems from the structure of the graph. Thus
\[ \sum n_i = n. \]

2. \( |Alg| = \sum f(A_i) \) stems from the definition.

3. \( f(A_i) \geq 2 \ast (2^{n_i-2} + n_i) \). This is a result of Proposition 4.1.2.

From the above facts we can get the following lower bound.

**Corollary 4.1.3** Any algorithm solving \( PC_\ell \) contains at least \( \frac{1}{2} \sum 2^{n_i} + 2n \) moves.

**Proof**: Let \( Alg \) be such algorithm. Then
\[ |Alg| = \sum f(A_i) \geq \sum 2 \ast (2^{n_i-2} + n_i) = \frac{1}{2} \sum 2^{n_i} + 2n. \]

**Theorem 4.1.4** The following algorithm \( Move_{PC_\ell}(n,S,D) \) is an optimal algorithm solving \( PC_\ell \), and it takes \( (\ell + r)2^{[\frac{\ell}{2}]} \) moves, \( r \equiv n \) (mod \( \ell \)).

**Algorithm 4.4** \( Move_{PC_\ell}(n,S,D) \)

1: Compute \( n_i, i = 1, 2, \ldots, \ell \)
\[
\begin{align*}
n_i = \begin{cases} 
\lceil \frac{n}{2} \rceil & \text{if } i \leq r \\
\lceil \frac{n}{2} \rceil & \text{else}.
\end{cases}
\end{align*}
\]

2: for \( i = 1 \) to \( \ell \) do
3: transfer \( n_i \) disks from \( S \) to \( C_i \) using \( Move_{H_1}(n, S, C_i) \).
4: end for
5: for \( i = \ell \) downto 1 do
6: transfer \( n_i \) disks from \( C_i \) to \( D \) using \( Move_{H_1}(n, D, C_i) \).
7: end for

**Proof**: The algorithm matches the bounds in Corollary 4.1.3, i.e. \( |Move_{PC_\ell}| = \frac{1}{2} \sum 2^{n_i} + 2n \). We need to proof that the chosen values for the \( n_i \) variables minimize the expression \( \frac{1}{2} \sum 2^{n_i} + 2n \).
First, let us prove that in each optimal algorithm \( \forall i \neq j, |n_i - n_j| \leq 1 \).
Let us assume to the contrary that \( \exists i \neq j \) such that \( |n_i - n_j| \geq 2 \), w.l.o.g. \( n_i > n_j \).
Therefore:
\[
2^{n_i} + 2^{n_j} = 2^{n_i - 1} + 2^{n_i - 1} + 2^{n_j} > 2^{n_i - 1} + 2 \cdot 2^{n_j} = 2^{n_i - 1} + 2^{n_j + 1}
\]
So by decrementing \( n_i \) by 1, and incrementing \( n_j \) by 1, we have a better algorithm, contradiction.

The constraint \( \forall i \neq j, |n_i - n_j| \leq 1 \), implies that an algorithm is optimal, if and only if, \( r \) arbitrary \( n_i \) variables get the value \( \lceil \frac{r}{7} \rceil \), while the rest of the variables get the value \( \lfloor \frac{r}{7} \rfloor \). Therefore, our algorithm is optimal. \( \square \)

We note that a variant of the \( PC_\ell \) graph, where vertices \( S \) and \( D \) are replaced with two SCCs of size three, is a special case of a Matrix-graph discussed in Chapter 5.

### 4.2 General Case

We study Hanoi graphs which are composed of \( \ell \) ”parallel” Hanoi graphs, for whom we know optimal algorithms. We provide optimal algorithms.

More formally: We are given \( \ell \) Hanoi graphs \( H_1, H_2, \ldots, H_\ell \), with \( \text{Source}(H_i) = S_i, \text{Destination}(H_i) = D_i \) and an optimal algorithm, \( \text{Opt}_i \) for \( H_i \), for each \( 1 \leq i \leq \ell \). We build the following graph \( H^{Par} = (V^{Par}, E^{Par}) \), where
\[
V^{Par} = \bigcup_{i=1, \ldots, \ell} V_i \cup \{S, D\}.
\]
Figure 4.2: Parallel Components Hanoi graph $H^{Par}$

$$E^{Par} = \bigcup_{i=1,\ldots,\ell} E_i \cup \{(S, S_i) : 1 \leq i \leq \ell\} \cup \{(D_i, D) : 1 \leq i \leq \ell\}.$$ 

See Figure 4.2 for an illustration.

Let $Alg$ be some algorithm solving $H^{Par}$, and let $A_i$ be the group of disks that resided in $H_i$, at some time during the algorithm. Where, $n_i = |A_i|$, $A = \{A_1, \ldots, A_\ell\}$, and $f_i(m) = |Opt_i(m)|$.

**Claim 4.2.1** Any algorithm solving $H^{Par}$, with the partition of $[1..n]$ as above, contains at least $\sum_{i=1}^{\ell} f_i(n_i) + 2 \cdot n$ moves.

**Proof:** $|Alg| = \sum |(Alg|_{A_i})| \geq \sum (f_i(n_i) + 2 \cdot n_i)$ this is due to definitions and the graph structure. \qed

We define the sequence $Opt'_i$ as $Opt_i$ with the following two changes:

1. Replace the first move $S_i \xrightarrow{j} v$ of disk $j$ with two consequent moves $S \xrightarrow{j} S_i$ and $S_i \xrightarrow{j} v$. 
2. Replace the last move \( v \xrightarrow{j} D \) of disk \( j \) with two consequent moves \( v \xrightarrow{j} D_i \) and \( D_i \xrightarrow{j} D \).

From definition, it is clear that \( |Opt'_i(n_i)| = f_i(n_i) + 2 \times n_i \). We also define \( \max_i \) as the maximal disk in \( A_i \), in a similar way we define \( \min_i \).

We focus on solutions in which each \( A_i \) consists of consecutive disks and \( \min_{i+1} = \max_i + 1 \).

**Claim 4.2.2** The following algorithm \( \text{Move}_{H_{Par}}(n, S, D) \) solves \( H_{Par} \) with \( \sum f_i(n_i) + 2 \times n \) moves.

**Algorithm 4.5** \( \text{Move}_{H_{Par}}(n, S, D) \)

1: for \( i = 1 \) to \( \ell \) do
2: \( \text{transfer} \ n_i \text{ disks from } S \text{ to } H_i \text{ using } Opt'_i[S \xrightarrow{\min} S_i, D_i \xrightarrow{\max} D]. \)
3: end for
4: for \( i = \ell \) downto 1 do
5: \( \text{transfer} \ n_i \text{ disks from } H_i \text{ to } D \text{ using } Opt'_i[D_i \xrightarrow{\max} D, D_i \xrightarrow{\min} D]. \)
6: end for

So, for a given division \( A \) of \([1..n]\), our algorithm is optimal. If we want to prove \( \text{Move}_{H_{Par}} \) to be optimal over all (legal) divisions, we need to minimize the following function:

\[
\min F(n_1, n_2, \ldots, n_\ell) = \sum_{i=1}^\ell f_i(n_i).
\]

with the following constraints:

1. \( n_i \geq 0 \) for each \( i = 1, \ldots, \ell \).
2. \( \sum_{i=1}^\ell n_i = n \).
We suggest two approaches for minimizing $F$. In Section 4.2.1 we use dynamic programming, and in Section 4.2.2 we use numeric approximation.

### 4.2.1 Discrete Solution

We provide a dynamic programming algorithm minimizing the function within $O(\ell n^2)$ steps, under the assumption that computing $f_i(m)$ for some $m$ takes $O(1)$ operations.

Let $mat[i, s]$ be the solution to the following sub-problem

1. $\min \sum_{j=1}^{i} f_j(n_j)$
2. $\sum_{j=1}^{i} n_j = s$.

We can define $mat[i, s]$ recursively in the following way.

$$mat[i, s] = \begin{cases} f_1(s) & \text{if } i = 1 \\ \min_{1 \leq \ell \leq s} \{ mat[i - 1, \ell] + f_i(s - \ell) \} & \text{otherwise.} \end{cases}$$

By using the standard techniques of dynamic programming, we achieve a bound of $O(\ell n^2)$ for computing the value of $mat[\ell, n]$, and the corresponding values for $n_1, \ldots, n_{\ell}$. They constitute the optimal solution sought for, by definition.

We note that it is desirable to find an algorithm computing the optimal division of $n$ in time polynomial in $k$ only. However, in the general case it is impossible, since the functions $f_i$ are assumed being arbitrary.
4.2.2 Numeric Approximation

In this section, we do not pretend to provide a well established approximation method, but rather a direction of study. We use the technique based on Lemma 2.4.2 to find an approximate optimal solution in $\mathbb{R}^\ell$. In case each $f_i$ is a continuously differentiable convex function, the point $(x_1, x_2, \ldots, x_k)$ satisfying $f_1'(x_1) = f_2'(x_2) = \ldots = f_k'(x_k)$, is the minimum of $F$ in $\mathbb{R}^\ell$. We can find that point, and round the results to get an approximate discrete solution. For example, consider a particular case, where the Hanoi graph is composed of two parallel components: a complete 3-peg graph $H_1$, and a complete 4-peg graph $H_2$. We assume correctness of the Frame-Stewart conjecture.

The time required to solve $H_1$ is

$$f_1(n_1) = 2^{n_1} - 1.$$ 

According to P.K. Stockmeyer [40], a good approximation function for the number of move required to solve $H_2$ is:

$$f_2(n_2) = 2\sqrt{2n_2}(\sqrt{n_2} - 1) + 1.$$
The minimum \((x_1, x_2)\) satisfies \(f'_1(x_1) = f'_2(x_2)\).

\[
f'_1(x_1) = \ln 2 \cdot 2^{x_1}, \quad (4.1)
\]

\[
f'_2(x_2) = 2\sqrt{2x_2} \left[ \frac{\ln 2}{\sqrt{2} x_2} (\sqrt{x_2} - 1) + \frac{1}{2\sqrt{x_2}} \right]
\]

\[
= 2\sqrt{2x_2} \left[ \frac{\sqrt{2} \ln 2 (\sqrt{x_2} - 1) + 1}{2\sqrt{x_2}} \right]
\]

\[
= 2\sqrt{2x_2} \left[ \frac{\sqrt{2} \ln 2}{2} + \frac{C}{\sqrt{x_2}} \right], \quad (4.2)
\]

where \(C > 0\) is some constant. From (4.1) and (4.2) we get

\[
x_1 = \sqrt{2x_2} + \log \left( \frac{\sqrt{2}}{2} + \frac{C}{\ln 2 \sqrt{x_2}} \right). \quad (4.3)
\]

We can approximate Equation (4.3) and get the pair of equations:

\[
x_1 \approx \sqrt{2x_2}
\]

\[
x_1 + x_2 = n
\]

and the solution for this quadratic equation is

\[
x_1 = \sqrt{2(n + 1)} - 1, \quad x_2 = n - \sqrt{2(n + 1)} + 1.
\]

We use ceiling and floor to round the results (we have only 2 options, so we choose the one yielding the lower value).
Chapter 5

Bounds for General Almost-Acyclic Hanoi Graphs

Chapter 5 has the following contents:

1. In Section 5.1, we give some local definitions.

2. In Section 5.2, consider general Hanoi graphs, such that it is possible to omit from each component a vertex, without changing $H^{SCC}$. We suggest algorithm solving them.

3. In Section 5.3, we prove our algorithm to be optimal within the family of block-algorithms, which use at most one vertex in each component for "storage".

4. In Section 5.4, we define a recurrence for a lower-bound on the number of steps required for solving general Hanoi graph.

5. In Section 5.5, we study "Matrix" graphs $H^{Mat}$, which are in a sense the "middle way", between "row-graphs" $PC_\ell$ and "column-graphs"
We consider them to be a kind of approximation for the general case. We provide an upper and a lower bound for them and show that this pair is tight.

### 5.1 Local Definitions

We study general Hanoi graphs $H$, within the family $AAG3_\ell$, with all SCCs isomorphic to $K_3$. We assume that $C(H)$ is numbered w.r.t. some topological sort of $H^{SCC}$ i.e., if there’s a directed path from $C_i$ to $C_j$ in $H^{SCC}$, then $i < j$. We denote the vertices of $C_i$ by $\{v_{i1}, v_{i2}, v_{i3}\}$. We also assume that $S = v_{i1}$ and $D = v_{i\ell}$.

We require the following property for Hanoi graphs in the considered class: Each component $C_i$ in $H$ contains a vertex $v_{ix}$, s.t. the removal of all the vertices $v_{ix}$, $i \in \{1, \ldots, \ell\}$, does not effect the graph $H^{SCC}$. More formally:

**Assumption 5.1.1** $\forall C_i \in C(H) \exists v_{ix} \in C_i$, s.t. $\tilde{H}^{SCC} = H^{SCC}$, where $\tilde{H} = H - \{v_{ix}\}_{C_i \in C(H)}$.

W.l.o.g. let us assume this vertex is $v_{i1}$, for each $i \in \{1, \ldots, \ell\}$. If there is a path from $v_{i1}$ to $v_{j1}$ in $H$, then there is a path from $C_i$ to $C_j$ in $H^{SCC}$, and by Assumption 5.1.1 there is a path from $\tilde{C}_i$ to $\tilde{C}_j$ in $\tilde{H}^{SCC}$. So, there is a path from $v_{i1}$ to $v_{j1}$ in $H$, which does not contain the vertices $v_{k1}$, for $i < k < j$. Let us denote this path by $p(i, j)$. We note that the possibility of transferring disks from $v_{i1}$ to $v_{j1}$ via $p(i, j)$, does not depend on whether
the vertices \( v_{k1}, i < k < j \), are free of disks or not. See Figure 5.1 for an example.

### 5.2 Description of the Algorithm

Our algorithm 5.2 \( \text{Move}_{\text{gen}}(H, 1, \ell) \) is parameterized by values \( k_{ij}(H) \). They are computed by Algorithm \( \text{CompPar}(H) \). In particular \( k_{i\ell}(H) \) is the number of blocks used by \( \text{Move}_{\text{gen}}(H, 1, \ell) \). We denote \( P(H_{ij}, C_h) = \min\{k_{ih}(H), k_{hj}(H)\} \) where \( h \in I(H_{ij}) \) and \( h \neq i, j \). We note that our algorithm has the property of Assumption 2.2.4(1), and that \( v(C_i) = v_{i1} \).

**Theorem 5.2.1** Algorithm \( \text{Move}_{\text{gen}}(H, r, q) \) is correct for any graph \( H \), satisfying assumption 5.1.1, and it transfers \( k_{rq} \) blocks from \( v_{r1} \) to \( v_{q1} \), having the properties of Assumption 2.2.4.
CompPar 5.1 \(k_{ij}(H)\)

1: \textbf{if} \(H_{ij} = \emptyset\) \textbf{then}
2: \hspace{1em} \textbf{return} 0
3: \textbf{else}
4: \hspace{1em} \text{\textit{Sum}} = 0
5: \hspace{1em} \textbf{for all} \(h \in I(H_{ij})\) \textbf{s.t.}, \(h \neq i, j\) \textbf{do}
6: \hspace{2em} \textit{Sum} + = \min\{k_{ih}(H), k_{hj}(H)\}
7: \hspace{1em} \textbf{end for}
8: \hspace{1em} \textit{Sum} + = 1.
9: \hspace{1em} \textbf{return} \textit{Sum}.
10: \textbf{end if}

Algorithm 5.2 \(Move_{gen}(H, r, q)\)

1: \textbf{if} \(\ell(H_{rq}) = 1\) or \(\ell(H_{rq}) = 2\) \textbf{then}
2: \hspace{1em} \text{Transfer one block from} \(v_{r1}\) \text{to} \(v_{q1}\) \text{using} \(BM_{(a)}\) or \(BM_{(b)}\) \text{accordingly.}
3: \textbf{else}
4: \hspace{1em} \textbf{for all} \(i \in I(H_{rq})\) \textbf{in the non-increasing order of} \(P(H_{rq}, C_i)\) \textbf{do}
5: \hspace{2em} \text{Transfer} \(P(H_{rq}, C_i)\) \text{blocks from} \(v_{r1}\) \text{to} \(v_{i1}\) \text{by} \(Move_{gen}(H, r, i)\).
6: \hspace{1em} \textbf{end for}
7: \hspace{1em} \text{Transfer one block from} \(v_{r1}\) \text{to} \(v_{q1}\) \text{via} \(p(r, q)\) \text{by using} \(BM_{(b)}\).
8: \hspace{1em} \textbf{for all} \(i \in I(H_{rq})\) \textbf{in the order of indices inverse to that in line 4 do}
9: \hspace{2em} \text{Transfer} \(P(H_{rq}, C_i)\) \text{blocks from} \(v_{i1}\) \text{to} \(v_{q1}\) \text{by} \(Move_{gen}(H, i, q)\).
10: \hspace{1em} \textbf{end for}
11: \textbf{end if}
Proof:

We denote by $M_1, M_2, \ldots M_{\ell-2}$ the ordered sets of components in line 4, i.e., $P(H, M_1) \geq P(H, M_2) \geq \ldots \geq P(H, M_{\ell-2})$.

It is easy to see that Algorithm 5.2 transfers $k_{1\ell}(H)$ blocks from $S$ to $D$, while having the properties of Assumption 2.2.4, if correct. We next show that the algorithm is correct. This is proved by induction on the number of components in the graph $\ell(H)$, or in short the graph size.

For $\ell(H) = 2$, $k_{12}(H) = 1$, and algorithm $BM_b$ (from Chapter 2) indeed transfers one block from $S$ to $D$.

Let us assume correctness for all graphs $H$ of size smaller than $\ell$. Let $C_i \in C(H)$ be some component s.t. $i \neq 1, \ell$. By the induction hypothesis, we can transfer in an empty graph, $k_{1i}(H_{1i})$ blocks from $S$ to $v(C_i)$. By definition, $P(H, C_i) \leq k_{1i}(H)$, hence we can transfer $P(H, C_i)$ blocks from $S$ to $v(C_i)$. A similar argument shows, that if we have $P(H, C_i)$ blocks stacked at $v(C_i)$, we can transfer them from there to $D$.

It will be convenient to encode certain sequences of moves of disks by paths in $H$. Thus $(m, p)$, where $p$ is the path $(e_1, e_2, \ldots, e_\ell)$, $e_i \in E$, signifies that disk $m$ first moves from the starting point of $e_1$ to the endpoint of $e_1$, then it moves from starting point of $e_2$ to the endpoint of $e_2$, and so forth.

We can look at the components of $H$ as a poset with the relation "$\leq$" defined as follows: $M_i \leq M_j$ iff there is a directed path from $v(M_i)$ to $v(M_j)$ in $H$. The relation $<$ is naturally defined as follows: $M_i < M_j$, iff $M_i \leq M_j$ and $M_i \neq M_j$.
but $M_i \neq M_j$.

**Lemma 5.2.2** Let $M_i, M_j$ be two components s.t. $M_i < M_j$, $M_i - v(M_i)$ and $M_j$ are free of disks and $\forall M_h, M_h < M_i$, $M_h \neq C_1$, $M_h$ is free of disks, then we can transfer $P(H, M_i)$ blocks from $S$ to $v(M_j)$.

**Proof:** By the conditions of the lemma, there exists a sequence $Alg$ to transfer $P(H, M_i)$ blocks from $S$ to $v(M_i)$. We will alter this sequence to transfer $P(H, M_i)$ blocks from $S$ to $v(M_j)$.

We build a new sequence $Alg'$, which is a simulation of $Alg$ with the destination $v(M_j)$ instead of $v(M_i)$, in the following way: For all block-moves from $v(C_h)$ to $v(C_i)$, with move-path $p$, replace $p$, with $(p(h, j) \cup (v_{js}, v_{jd}))$, where $v_{js}$ is the last vertex in $p(i, j)$, and $v_{jd}$ is the last in $p$. Replace every disk move $(m, v_{ix}, v_{iy})$, with $(m, v_{ix}, v_{jy})$.

From the construction of $Alg'$, it is clear that all the block-moves in $Alg$ with destination in $M_i$, are replaced by legal block-moves with destination in $M_j$. We note that in the end of $Alg'$ all disks are gathered on $v(M_j)$, since in $Alg$ they are gathered on $v(M_i)$, and $Alg'$ mirrors $Alg$ actions from $M_i$ to
Thus, $\text{Alg}'$ transfers $P(H, M_i)$ blocks from $S$ to $v(M_j)$. \hfill \Box$

We call a component $M_i$ which contains $P(H, M_i)$ blocks at $v(M_i)$ a filled component. The following observation is straightforward.

**Observation 5.2.3** All components in $H$ except for $C_1$ and $C_\ell$, just after the execution of line 5 in Algorithm 5.2, are either filled or empty of disks.

The idea behind the rest of the correctness proof is that at any stage of the algorithm, disks residing in filled components (other components are empty) do not obstruct other components from becoming filled.

We prove possibility of lines 4-5 of Algorithm 5.2 by induction on the for-loop in those lines.

**Basis:** $i = 1, 2$.

1. For $i = 1$, the graph is empty, so we can transfer $P(H, M_1)$ blocks from $S$ to $v(M_1)$ by the induction hypothesis.

2. For $i = 2$, we distinguish the following cases:

   - **Case 1:** $M_2 \leq M_1$ or $M_2$ is not comparable with $M_1$. Then, $M_1$ does not belong to any path to $M_2$, so we transfer $P(H, M_2)$ blocks from $S$ to $v(M_2)$, as in the empty graph.

   - **Case 2:** $M_1 \leq M_2$. We note that $P(H, M_1) \geq P(H, M_2)$, and $\forall M_h$ s.t. $M_h < M_1$, $M_h$ is free of disks. Therefore Lemma 5.2.2 enables us to transfer $P(H, M_2)$ blocks from $S$ to $v(M_2)$.

**Induction step:** assuming we filled $i - 1$ components, let us prove that we can fill $M_i$ also.
1. We distinguish the following two cases.

   Case 1: ∄ \( M_j \in \{M_1, \ldots, M_{i-1}\} \) s.t. \( M_j \leq M_i \). So we transfer \( P(H, M_i) \) blocks from \( S \) to \( v(M_i) \), since \( H[C_1, M_i] \) is empty, except for \( C_1 \).

   Case 2: \( \exists M_j \in \{M_1, \ldots, M_{i-1}\} \) s.t. \( M_j \leq M_i \). Let \( M_{j_0} \) be the minimal component in \( \{M_1, \ldots, M_{i-1}\} \) with respect to ”\( \leq \)”, which satisfies \( M_{j_0} \leq M_i \). We notice that by definition \( P(H, M_{j_0}) \geq P(H, M_i) \). Thus, Lemma 5.2.2 enables us to transfer \( P(H, M_i) \) blocks from \( S \) to \( v(M_i) \).

After the execution of lines 4-5, all the components in \( H \), except \( C_1, C_\ell \), are filled. Line 7 is possible by using \( BM(b) \), extended via the path \( p(1, \ell) \).

We say, we emptied \( M_i \) for some \( i \in \{1, \ldots, \ell - 2\} \), if we transferred all the blocks in it to \( D \). To prove possibility of lines 8-9, we make use of the following observation, which is valid by the symmetry of Algorithm 5.2.

**Observation 5.2.4** For all \( i \in \{1, \ldots, \ell - 2\} \), the configurations just after executing line 5 and just before executing line 9 are identical, except for \( C_1 \) and \( C_\ell \).

Since the only difference in the configurations, is the placement of some set of the largest blocks in \( C_\ell \) instead of \( C_1 \), which does not obstruct moves of disks in other blocks, we regard those configurations as identical. Due to the above observation, it is possible to denote \( \forall i \in \{1, \ldots, \ell - 2\} \), the configuration just after filling \( M_i \) or just before emptying it as \( Conf_i \).

Let \( M_i \) be some component s.t. \( 1 \leq i \leq \ell - 2 \). To prove correctness of lines 8-9, it is sufficient to show that there exists a sequence to empty it,
beginning from the configuration $Conf_i$, and keeping configurations at other components.

Let us denote by $Alg$ the application of Algorithm 5.2 on $H$ and by $\overline{Alg}$, the application of Algorithm 5.2 on $\overline{H}$, with $Source(\overline{H}) = D$ and $Destination(\overline{H}) = S$.

Note that for each $C_j \in C(H) \setminus \{C_1, C_\ell\}$, $P(H, C_j) = P(\overline{H}, C_j)$. Therefore, applying $\overline{Alg}$ in the reverse order, just after $\overline{Alg}$ filled $M_i$, i.e., in configuration $Conf_i$, empties $M_i$, and keeps configurations at other components.

Thus, we have proved correctness of lines 8-9, and hence correctness of the algorithm. So, Algorithm 5.2 is legal and it transfers $k_{1\ell}(H)$ blocks from $S$ to $D$.

\section{Optimality Proof in the Class of Block-Algorithms}

In this Section, we prove that the parameter $k_{1\ell}(H)$, computed in the previous section, is the maximal number of blocks we can transfer from $v(C_1)$ to $v(C_\ell)$ in the class of block-algorithms.

\textbf{Lemma 5.3.1} Every Hanoi-graph $H$, has an optimal solution in which the largest block moves only once.

\textbf{Proof:} Suppose a block-algorithm solving $HAN(H, n)$ is given, in which the first block-move of the largest block, $B_k$, is from $S$ to $v(C_j)$. At the stage when this block-move is carried out, $C_\ell$ is empty. Since otherwise, just
after $B_k$ moves later to $C_\ell$, $C_\ell$ contains (at least) two blocks on two different pegs, contradicting Assumption 2.2.4. So, we can change the given solution as follows. Replace the first block-move of $B_k$ by $BM(c)$ from $S$ to $D$ via $p(1, \ell)$, and continue with all block-moves of the original algorithm, omitting block-moves of $B_k$.

Notice that, up to the first block-move of $B_k$, the two sequences yield the same configurations of blocks on $H$. From that point on, the only difference between the configuration might be the location of $B_k$. Since $B_k$ is largest, moves of other blocks can be performed without any obstruction.

In view of the lemma, we may restrict ourselves to solutions of $HAN(H, n)$ in which the largest block moves but once.

**Theorem 5.3.2** The maximal number of blocks we can transfer from $S$ to $D$ in Hanoi graph $H$ using block-algorithm, which satisfies Assumption 2.2.4 is $k_1\ell(H)$.

**Proof:** Let us denote by $k(H, C_i, C_j)$ the maximal number of blocks that we can transfer from $v_{i1}$ to $v_{j1}$ under the restrictions of Assumption 2.2.4 in an empty graph.

We want to prove that $k(H, C_i, C_j) = k_1\ell(H)$. Inequality in one direction is implied by the correctness of our algorithm. The inequality in the other direction is proved by induction on the graph size.

By Claim 2.3.2, for $\ell = 2$, we can transfer at most one block from $S$ to $D$, so the proposition is correct. Let $\ell \geq 3$, and let us assume correctness of the proposition for all Hanoi-graphs of size smaller than $\ell$. Consider
a block-algorithm solution for $H$ satisfying the property in Lemma 5.3.1. From the induction assumption $k(H_{1i}, C_i, C_i) = k_{1i}(H_{1i}) = k_{1i}(H)$, similarly $k(H_{i\ell}, C_i, C_{\ell}) = k_{i\ell}(H)$. Thus, the number of blocks residing at $C_i$, $i \neq 1, \ell$, at the stage when the largest block is moved from $C_1$ to $C_{\ell}$ does not exceed $\min\{k_{1i}(H), k_{i\ell}(H)\} = P(H, C_i)$. Hence the total number of blocks $k(H, C_1, C_{\ell})$ satisfies:

$$k(H, C_1, C_{\ell}) \leq \sum_{C_i \in \mathcal{C}(H), i \neq 1, \ell} P(H, C_i) + 1.$$ 

\[ ]

5.4 General Lower Bound

In this section we show a recursive formula for computing a lower bound on the number of moves required by any algorithm to solve $HAN(H, n)$.

**Theorem 5.4.1** The minimal number of steps of any algorithm solving $HAN(H, n)$ is $2^n \pi^{\ell/2} - c(\ell)^2$, for some function $c(\ell)$, where $g(H) = g(H_{1\ell})$ is defined by Recurrence 5.3.

**Recurrence 5.3** $g(H_{ij})$

1: if $i = j$ then
2: \hspace{1cm} $g(H_{ij}) = 1$
3: else
4: \hspace{1cm} $g(H_{ij}) = 3 \sum_{h \in I(H_{ij})} \min\{g(H_{ih}), g(H_{hj})\} + 2.$
5: end if

**Proof:** The proof is by induction on $\ell$.

**Basis:** $\ell = 1, 2$. 
For \( \ell = 1 \) we have the classic problem, \textit{Recursive3Peg}, requiring \( 2^n - 1 \) steps to solve, which accords \( g(H) = 1 \). For \( \ell = 2 \), in the recurrence the minimum on the empty set is set to zero. Let us look at the configuration of \( D_n \setminus \{n\} \) during the first move of disk \( n \). Let us denote the number of disks in \( C_1 \) by \( q \), and thereby the number of disks in \( C_2 \) is \( n - q - 1 \).

From Claim 2.3.3, the minimal number of moves required to move the \( q \) disks in \( C_1 \) before the first move of disk \( n \), is at least \( 2^{q-1} \). In a similar way the minimal number of moves required to move the \( n - q - 1 \) disks in \( C_2 \), after the last move of disk \( n \) is \( 2^{n-q-2} \). Altogether we have \( 2^{q-1} + 2^{n-q-2} \) moves. A simple calculation shows that the minimal value of this function is \( 2^{\lfloor \frac{n-3}{2} \rfloor} + 2^\lceil \frac{n-3}{2} \rceil \leq 2^n \).

\textit{Induction Step:} For \( \ell \geq 3 \), assume the correctness of the claim for all values of \( \ell' < \ell \) and for all \( n \), and prove it for \( \ell \) and for all \( n \).

Let us look at the configuration of \( D_n \setminus \{n\} \) during the first move of disk \( n \). Let us denote by \( n_i \) the number of disks in \( C_i \), for each \( i \in \{1, \ldots, \ell\} \). Note that \( \sum_{i=1}^{\ell} n_i = n - 1 \). From Claim 2.3.3, the minimal number of moves required to transfer \( n_1 \) disks from \( v_{11} = S \), to some configuration in \( C_1 \) is \( 2^{n_1-1} \). From symmetry, the minimal number of moves required to transfer \( n_\ell \) disks from some configuration in \( C_\ell \), to \( v_{\ell1} = D \) is \( 2^{n_\ell-1} \).

From the induction hypothesis, for \( 2 \leq i \leq \ell - 1 \), the minimal number of steps required to move \( n_i \) disks from \( S \) to \( v_{i1} \in C_i \), is at least \( 2^{\frac{n_i}{H_{11}}-c_1(i)} \), for some function \( c_1(i) \). Similarly, the minimal number of moves required to move \( n_i \) disks from \( v_{i1} \in C_i \) to \( D \) is at least \( 2^{\frac{n_i}{H_{i\ell}}-c_2(i)} \), for some function
Using Lemma 3.3.2, we get that for \( i \in \{2, \ldots, \ell - 1\} \), the minimal number of moves required to move \( n_i \) disks from \( S \) to \( C_i \) and from \( C_i \) to \( D \) is:

\[
2^{2^{g(H_{1i})} - c_1(i)} + 2^{2^{g(H_{1i})} - c_2(i)} \geq 2^{\min\{g(H_{1i}), g(H_{1\ell})\} - c(i)},
\]

where \( c(i) = \max\{c_1(i), c_2(i)\} \).

Altogether, we get from Proposition 2.4.3 (1) the following lower bound:

\[
\sum_{i=2}^{\ell-1} 2^{\min\{g(H_{1i}), g(H_{1\ell})\} - c(i)} + 2^{n_1 - 1} + 2^{n_\ell - 1} \geq 2^{\min\{g(H_{1i}), g(H_{1\ell})\} + \log(g(H_{1\ell}))} \geq 2^{\frac{n}{g(H_{1\ell})} - c(\ell)},
\]

where

\[
s(\ell) = \sum_{i=2}^{\ell-1} m_i (\log m_i - c(i)) - 2, \quad m_i = \min\{g(H_{1i}), g(H_{1\ell})\},
\]

\[
c(\ell) = \frac{s(\ell) + 1}{g(H_{1\ell})} + \log(g(H_{1\ell})).
\]

We note that our pair of recurrences for upper and lower bounds are similar to those in Chapter 3, which we proved to form a tight pair. We believe our pair is also tight, but could not prove it.

### 5.5 Matrix Graph

In this section we study Matrix graphs. Matrix graph \( H_{hw}^{\text{mat}} \) is composed of \( h \) rows and \( w \) columns of components, and two separate components for the source and destination. For each component, there are edges between all it’s
vertices and all the vertices belonging to "lower" rows. See Figure 5.3 for an illustration. A formal definition is as follows:

\[ \mathcal{C}(H_{\text{mat}}^{\text{mat}}) = \{C_{i,j} \cup C_{0,1} \cup C_{h+1,1} : 1 \leq i \leq h, 1 \leq j \leq w\} \]

\[ E(H_{\text{mat}}^{\text{mat}}) = \{(u, v) : (u, v) \in C_{i,j} \text{ or } (u \in C_{i,j}, v \in C_{f,k}, \text{ where } i < f)\}. \]

\[ S = v_{(0,1),1} \text{ and } D = v_{(h+1,1),1}. \]

Matrix graphs may be considered as intermediate between "column" graphs \( H_{\ell}^{TC} \), studied in Chapter 3, and "row" graphs \( PC_\ell \), studied in Chapter 4.
5.5.1 Lower Bound

Let us denote \( g(H_{hw}^{\text{mat}}) \) by \( b_{hw} \), where function \( g \) is defined in Theorem 5.4.1.

**Lemma 5.5.1** \( b_{0w} = 2 \), and for each \( h \geq 1 \), \( b_{hw} = b_{h-1,w} + 3w \cdot b_{\lfloor \frac{h}{2} \rfloor - 1,w} \).

**Proof:** For \( b_{0w} \) we get the graph \( H_2^{\text{max}} \), which satisfies \( g(H_2^{\text{max}}) = 2 \), by Proposition 3.3.4. For \( b_{1w} \), using Recurrence 5.3, \(^1\) we get:

\[
b_{1w} = g(H_{1w}^{\text{mat}}) = 3 \sum_{j=1}^{w} \min\{g(H_{(0,0),(1,j)}), g(H_{(1,j),(2,1)})\} + 2 = 3w \sum_{j=1}^{w} \min\{b_{0w}, b_{0w}\} + 2 = 3w \cdot b_{0w} + 2.
\]

For general \( h \geq 2 \), by the graph structure, we get:

\[
b_{h} = g(H_{hw}^{\text{mat}}) = 3 \sum_{i=1}^{h} \sum_{j=1}^{w} \min\{g(H_{(0,0),(i,j)}), g(H_{(i,j),(h+1,1)})\} + 2 = 3w \sum_{i=1}^{h} \min\{b_{i-1}, b_{h-i}\} + 2 \tag{5.1}
\]

\[
= \begin{cases} 
6w \sum_{i=0}^{\lfloor \frac{h}{2} \rfloor - 1} b_i + 2 & h \equiv 0 \pmod{2}, \\
6w(\sum_{i=0}^{\lfloor \frac{h}{2} \rfloor - 1} b_i + b_{\lfloor \frac{h}{2} \rfloor - 1}) + 2 & h \equiv 1 \pmod{2}.
\end{cases}
\]

We can show the correctness of our claim from (5.1) by using a technique similar to the one used in the proof of Fact 3.3.3. \( \square \)

\(^1\)In Recurrence 5.3 we assume each component has one index, e.g., \( C_1, C_2, \) etc., while in \( H_{hw}^{\text{mat}} \) we use double index notation. We assume implicit conversion function \( f(i,j) = (i-1) \cdot w + j \), for the indices when necessary.
Proposition 5.5.2 There exists a constant $C$, such that $b_{hw} \leq Cw^{\lceil \log h \rceil + 1}h^{\frac{1}{2}\log h}$ for any $w, h \geq 1$.

Proof: For $h = 1$ we get from Lemma 5.5.1

$$b_{1w} = 6w + 2 \leq 7w = 7w^{\lceil \log 1 \rceil + 1}1^{\frac{1}{2}\log 1}.$$ 

General $h \geq 2$. In Chapter 3 we analyzed the recurrence $(a_h)_{h=2}^\infty$

$$a_2 = 2,$$

$$a_h = a_{h-1} + 3a_{\lceil \frac{h}{2} \rceil}, \quad h \geq 3,$$

and showed in Theorem 3.3.1 (2) that $a_h \leq C_2h^{\frac{1}{2}\log h}$ for any $h \geq 2$, where $C_1$ is some constant. On the other hand from Lemma 5.5.1, we get:

$$b_{2w} = b_{1w} + 3w \cdot b_{0w}$$

$$= 3w \cdot b_{0w} + 2 + 3w \cdot b_{0w}$$

$$= 12w + 2.$$ 

$$b_{hw} = b_{h-1,w} + 3w \cdot b_{\lceil \frac{h}{2} \rceil - 1,w}$$

$$\leq b_{h-1,w} + 3w \cdot b_{\lceil \frac{h}{2} \rceil - 1,w}, \quad h \geq 3.$$ 

Applying Lemma 2.4.1 with $(a_h)_{h=2}^\infty$, $(b_h)_{h=2}^\infty$, $k = 3$ and $\lambda = w$, we get

$$b_{hw} \leq \frac{b_2}{a_2}w^{\lceil \log h \rceil}a_h \leq C_1(6w + 1)w^{\lceil \log h \rceil}h^{\frac{1}{2}\log h}$$

$$\leq 7C_1w^{\lceil \log h \rceil + 1}h^{\frac{1}{2}\log h}.$$ 

Summarizing, the statement holds for $C = \max\{7, 7C_1\}$ in the entire range of parameters. \qed
5.5.2 Lower Bound Properties

Let us study the behavior of the lower-bound of Proposition 5.5.2 as parameterized by $\alpha \in [0, 1]$, so that $h = \ell^\alpha$ and $w = \ell^{1-\alpha}$. We start by analyzing $b_{hw}$ for specific values of $\alpha$, then we study $b_{hw}$ for general $\alpha$. We finish by analyzing $b_{hw}$ for small values of $h$.

By Proposition 5.5.2,

$$b_{hw} \leq C(\ell^{1-\alpha})^\lceil \log \ell^{\alpha} \rceil + 1 (\ell^\alpha)^{\frac{1}{2}} \log \ell^\alpha$$

$$= C\ell^{(1-\alpha)(\lceil \alpha \log \ell \rceil + 1) + \frac{\alpha^2}{2} \log \ell}.$$  \hspace{1cm} (5.2)

For the two boundary and the middle pairs $(w, h)$ we have:

- $\alpha = 0 \Rightarrow w = \ell, h = 1,$

$$b_{h,w} \leq C\ell.$$

- $\alpha = \frac{1}{2} \Rightarrow w = \sqrt{\ell}, h = \sqrt{\ell},$

$$b_{h,w} \leq C\ell^{\frac{1}{2}}((\lceil \frac{1}{2} \log \ell \rceil + 1) + \frac{1}{2} \log \ell) < C\ell^{\frac{1}{2}}((\frac{1}{2} + \frac{1}{2}) \log \ell + 2) = C\ell^{\frac{1}{2} \log \ell + 1}.$$

- $\alpha = 1 \Rightarrow w = 1, h = \ell,$

$$b_{h,w} \leq C\ell^{\frac{1}{2} \log \ell}.$$

For the study of $b_{hw}$ for general values of $\alpha$, we replace inequality (5.2) with the following expression.

$$b_{hw} < C\ell^{(1-\alpha)(\alpha \log \ell + 2) + \frac{\alpha^2}{2} \log \ell}$$

$$= C\ell^\alpha(1-\frac{\alpha}{2}) \log \ell + 2(1-\alpha).$$  \hspace{1cm} (5.3)
See Figure 5.4 for the behavior of the coefficient of $\log \ell$ in the exponent of equation (5.3).

**Claim 5.5.3** Given the number of components, $\ell + 2$, where $\ell \geq 6$, the Matrix-graph $H_{hw}^{Mat}$ with the minimal lower-bound (5.3) is $H_{1}^{Mat}$, i.e., $H_{\ell+2}^{\max}$.

**Proof:** For $\ell \geq 6$, the expression $\alpha (1 - \frac{\alpha}{2}) \log \ell + 2(1 - \alpha)$ has a positive derivative for $0 \leq \alpha \leq 1$. Thus, the right side of inequality (5.3) gets its maximal value for $\alpha = 1$. \hfill $\Box$

**Proposition 5.5.4** For small values of $h$, such that $\log h \ll \log \ell$, the lower bound is of order of magnitude $\sim \ell^{\log h + 1}$.

**Proof:** Substituting $w = \left( \frac{\ell}{h} \right)$, in Proposition 5.5.2 yields the following lower
bound:
\[
b_{hw} \leq C \left( \frac{\ell}{h} \right)^{\lceil \log h \rceil + 1} h^\frac{1}{2} \log h
\]
\[
= C^{\ell \lfloor \log h \rfloor + 1} \cdot h^\frac{1}{2} \log h - (\lfloor \log h \rfloor + 1)
\]
\[
= C^{\ell \log h + 1} + \log^2 h - (\lfloor \log h \rfloor + 1) \cdot \log h
\]
\[
\approx C^{\ell \log h + 1}.
\]

\[\square\]

5.5.3 Algorithm

It is not hard to see that \( P(H_{hw}^{\text{Mat}}, C_{ij}) \) gets its maximal value for \( i = \lceil \frac{h}{2} \rceil, \)
\( 1 \leq j \leq w. \) So applying Algorithm 5.2 to \( H_{hw}^{\text{Mat}} \) yields Algorithm 5.4.

Algorithm 5.4 \text{MoveMat}(H_{hw}^{\text{Mat}}, S, D)

1: if \( h = 0 \) then
2: transfer one block from \( S \) to \( D. \)
3: else
4: for \( i \leftarrow 1 \) to \( w \) do
5: \text{MoveMat}(H_{(0,0),([\frac{h}{2}],i)}, S, v([\frac{h}{2}],i)).
6: end for
7: \text{MoveMat}(H_{hw}^{\text{Mat}} \setminus \bigcup_{i=1}^{w} C_{[\frac{h}{2}],i}, S, D).
8: for \( i \leftarrow w \) downto 1 do
9: \text{MoveMat}(H_{([\frac{h}{2}],i),(h+1,1)}, v([\frac{h}{2}],i), D).
10: end for
11: end if

Denoting the number of blocks Algorithm 5.4 transfers in \( H_{hw}^{\text{Mat}} \) from \( S \)
to \( D \) by \( c_{hw} \), we get the following recursion.

Lemma 5.5.5 \( c_{0w} = 1 \) and for each \( h, w \geq 1, c_{hw} = c_{h-1,w} + w \cdot c_{[\frac{h}{2}],-1,w}. \)
We can use the technique used in Theorem 3.2.1 (2) with slight modifications to get a similar result for $c_{hw}$.

**Lemma 5.5.6** For any $\varepsilon > 0$, there exists a constant $C_{\varepsilon} > 0$, such that $c_{hw} \geq C_{\varepsilon} \cdot w \cdot h^{\left(\frac{1}{2} - \varepsilon\right) \log h}$ for each $h \geq 1$.

### 5.5.4 Measuring the Tightness of Our Bounds

Given a Matrix graph $H_{n,w}^{mat}$, we get from Lemma 5.5.1 a lower bound on the number of moves required to solve $HAN(H, n)$ i.e. $\Omega(2^n_{\frac{1}{2}w})$. On the other hand we introduced a block-algorithm solving the problem with $O(2^n_{\frac{1}{2}w})$ steps. We show in this section that $b_{hw}$ and $c_{hw}$, are in fact, very close to each other.

**Proposition 5.5.7** For each $h \geq 0$, $\frac{b_{hw}}{c_{hw}} \leq 2 \cdot 3^{\left\lceil \log h \right\rceil} < 6 \cdot h^{\log 3}$.

**Proof:** For $h = 0$, $b_{0w} = 2$, whereas $c_{0w} = 1$. So, $\frac{b_{0w}}{c_{0w}} = 2 = 2 \cdot 3^0$.

For $h \geq 1$,

\[
\begin{align*}
    b_{hw} &= b_{h-1,w} + 3w \cdot b_{\left\lceil \frac{h}{2} \right\rceil - 1,w}, \\
    c_{hw} &= c_{h-1,w} + w \cdot c_{\left\lceil \frac{h}{2} \right\rceil - 1,w}.
\end{align*}
\]

Applying Lemma 2.4.1 with slight modifications yields:

\[
\frac{b_{hw}}{c_{hw}} \leq 2 \cdot 3^{\left\lceil \log h \right\rceil} < 2 \cdot 3^{\log h + 1} = 6 \cdot h^{\log 3}.
\]

\[\square\]

We note that $6 \cdot h^{\log 3} < 6 \cdot h^{1.6}$. That is, the ratio is a low degree polynomial depending on $h$, which is of a lesser order then the order of magnitude of $c_{hw}$ and $b_{hw}$. 
Bibliography


