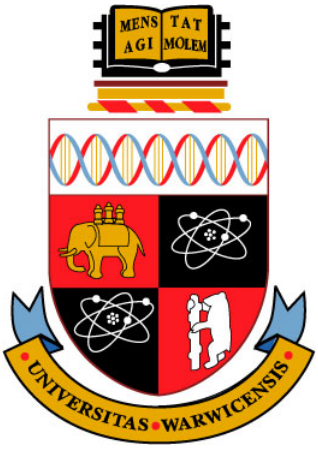

An analysis of strategies in the re-pairing game

CS344 Discrete Mathematics Project

Laveen Chandnani

Supervisors: Dr. Dmitry Chistikov & Dr. Matthias Englert

Department of Computer Science

Abstract

The project expands on the work of Chistikov and Vyali, which introduced a simple one-player game; The re-pairing game can be played on any well-formed sequence of opening and closing brackets (a Dyck word). A move consists of "pairing" any opening bracket with any closing bracket to the right of it, and "erasing" the two. The process is repeated until we are left with no remaining brackets. Such a game can have many strategies, but the effectiveness of a strategy is measured by it's width, which is the maximum number of nonempty segments of symbols seen during a play of the game.

Keywords: *Dyck language, Re-pairing brackets, Combinatorics, Web application, Python, ReactJS*

Acknowledgements

I'd like to thank my dissertation supervisors, Dr. Dmitry Chistikov and Dr. Matthias Englert, for their invaluable guidance, support and feedback throughout this project.

I'd also like to thank Melany Henot, Brendan Bell, Raumaan Ahmed, Varun Chodanker and Devon Connor for engaging in insightful discussions on some of the problems I tackled, and for their continued support which enabled me to complete this project throughout a difficult year.

Contents

1 Introduction

We start with a simple one-player game. Take any Dyck word (a balanced sequence of opening and closing brackets), for example:

```
(()())
```

A **move** in this game consists of pairing any opening bracket with any closing bracket to its right, and replacing the two with a blank space (denoted with the _ symbol). This process is repeated until there are no more brackets to pair up, resulting in the empty string, and the sequence of moves made is called a **re-pairing**. An example of a re-pairing is as follows:

```
(()())
(_)(_)
___(_)
______
```

Note that our first move pairs up two brackets that are not matched to each other in the initial word. We allow such moves in a re-pairing.

Finding any re-pairing following these rules is a trivial task, but we want to measure how good a re-pairing is. For this we introduce a property called the **width**, defined as the maximum number of non-empty segments of brackets seen during the re-pairing. We can see that the previous re-pairing had width 2. A natural question then arises; can we obtain a re-pairing with a smaller width? The answer for this particular string is yes, and such a re-pairing is shown below:

```
(()())
_()()_
_()___
______
```

Here we see this re-pairing has width 1. Our goal is to try and minimise this width as much as possible.

1.1 Objectives

This project aims to take both a theoretical and practical approach to this game. It is split into two parts; a literature review and software development.

The literature review aims to assess the current knowledge available on the game by understanding and expanding on the relevant background knowledge, context and results. This will also help gain a more intuitive grasp on the topic.

The software development aspect aims to implement strategies in the re-pairing game to allow for visualisation and experimentation. A subgoal of this is to attack the goal of minimising width with software, by analysing certain strategies and subcases of Dyck words.
More specifically, the objectives of the project can be outlined as follows:

1. An introduction to the relevant definitions and content to lay out background knowledge.

2. Expanding on and providing potential novel insights on well established results.

3. Establishing the research landscape on the re-pairing game and relevant topics.

4. Creating software to demonstrate re-pairing strategies from literature and allow manual experimentation on Dyck word re-pairings.

5. Using software to analyse subcases of Dyck words and exhaustively find their width in an attempt to pin down the formula of a general Dyck word.

1.2 Related work

This project builds on the work from Chistikov and Vyalyi [1], whose paper serves as a basis for this project. The paper introduces this game as a way to prove a lower bound on translating one-counter automata (OCA) into Parikh-equivalent nondeterministic finite automata (NFA). It provides lower bounds on the width for simple and non-simple re-pairings, which is then used to prove lower bounds on these translations.

Due to the specialised nature of this game's origins, there are no other works on this game at the time of writing.

2 Literature Review

This section aims to cover the relevant definitions, theorems, and results from literature. We will also attempt to provide insight on and formalise some well known ideas in the context of this project.

2.1 Background

2.1.1 Dyck Words

Definition. *A **Dyck Word** is a string that satisfies the following:*

1. *The only characters in the string are "(" or ")".*
2. *At any point in the string, there are no more "(" brackets than ")" brackets.*
3. *The string contains the same number of "(" and ")" brackets*

We can alternatively define a Dyck word through the convention "(" $= 1$ and ")" $= -1$:

Definition ([1])**.** *A **Dyck word** is a sequence $D = (d_n)_{n\geq 1}$ such that $d_n \in \{1, -1\}$ and the following is satisfied:*

1. *For every $1 \leq k \leq n$, we have $\sum_{i=1}^{k} d_i \geq 0$*
2. *$\sum_{i=0}^{n} d_i = 0$*

2.1.2 North-East Lattice Walks

These words have interesting combinatorial properties, one of which is their unsurprising link to Lattice Walks:

Definition. *A **North-East lattice walk** is a sequence of vectors (a, b) starting at $(0, 0)$ and finishing at (n, n), such that every step on the walk goes from some (a, b) to either $(a + 1, b)$ or $(a, b + 1)$.*

In other words, every step is exactly one unit vector right or one unit vector up.

Theorem. *Every Dyck word of length $2n$ corresponds to a unique North-East lattice walk from $(0, 0)$ to (n, n) that does not cross above the line $y = x$.*

This is a trivial mapping, as we can take a move east to be "(", and a move north to be ")". This works out as expected since we have $2n$ brackets to match, and n moves east and north each during such lattice walks. Our restriction on Dyck words that ensures there are never more ")" brackets than "(" brackets corresponds to the restriction of the walk being below (or on) the line $y = x$. This connection gives us another result:

Theorem ([2])**.** *The number of Dyck words of length* $2n$ *corresponds to the* n^{th} *Catalan number* $C_n = \sum_{i=1}^{n-1} C_i C_{n-1-i}$.

Proof. Let W_n be the number of North-East lattice walks from $(0, 0)$ to (n, n) that do not cross above the line $y = x$. Given any such walk, it may touch this line at any (i, i) where $0 \leq i \leq n - 1$. This gives us a natural recursion; every such North-East lattice walk can be broken down into a North-East lattice walk from $(0, 0)$ to (i, i), and from (i, i) to (n, n). Using this idea, we see that $W_n = \sum_{i=0}^{n-1} W_i W_{n-1-i}$. This is precisely the recursion for the n^{th} Catalan number. □

2.1.3 Binary Trees

However, we can also relate this result to the concept of binary trees as well. Suppose we had four factors of a number $x = x_1x_2x_3x_4$. How many ways can we *completely parenthesize* this expression of factors (i.e. place brackets around this to group factors)?

1. $(x_1(x_2x_3))x_4$
2. $((x_1x_2)x_3)x_4$
3. $x_1(x_2(x_3x_4))$
4. $x_1((x_2x_3)x_4)$
5. $(x_1x_2)(x_3x_4)$

There are 5 ways, but this is the same as $C_3 = \frac{1}{3+1}\binom{2(3)}{3} = 5$. To understand why, let G_n be the number of ways to place brackets around $x = x_1 \dots x_n$. Note that the outermost factor will not be within any pair of brackets.

We look at the leftmost "(", and its matching ")". Suppose this surrounds $0 \leq i \leq n - 1$ factors. We can completely parenthesize this expression of factors in G_i ways. Since the outermost factor will not be within any pair of brackets, this leaves a further $n - i - 1$ factors to completely parenthesize, which can be done in G_{n-i-1} ways. Therefore the number of ways to completely parenthesize an expression of n factors is $G_n = \sum_{i=0}^{n-1} G_i G_{n-1-i}$, which is precisely the n^{th} Catalan number.

Theorem. *The number of proper binary trees with $n+1$ leaves is C_n.*

To prove this, we first require a lemma.

Proof. Recall that a proper binary tree is a tree such that every non-leaf node has 2 nodes. Let P_n be the number of proper binary trees with n leaf nodes. We see that □

References

[1] D. Chistikov and M. Vyalyi, "Re-pairing brackets," in *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pp. 312–326, 2020.

[2] J. Rukavicka, "On generalized dyck paths," *The Electronic Journal of Combinatorics*, vol. 18, 2013.