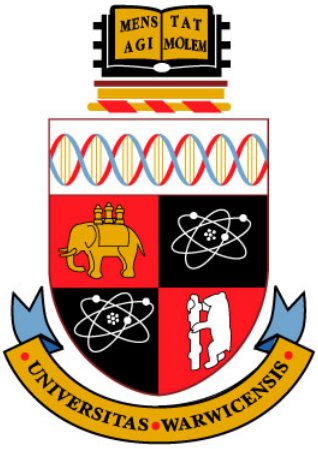

An analysis of strategies in the re-pairing game

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 aims to assess the current literature available on the game and expand on their results. This allows us to get a broader picture of the research landscape on the game, as well as a deeper and more intuitive grasp on certain results. The project also aims to implement these strategies in a practical way that allows for visualisation and experimentation. Finally, the project aims to attack the goal of minimising width with a practical approach, 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 relevant 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 Background

3 Literature Review

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.