Determining the Complexity of Parallel Circuits: A Proposal

Brian Lee

May 2017

Abstract

The goal of the experiment is to determine if Parallel Circuits are NP-Complete or P-Complete. For this purpose, parallel circuits will be augmented with switches, which will be modelled in a computer program's logic flow. To start, three parallel circuits are to be constructed, consisting of 1,2, and 3 resistors respectively. From there, induction will be utilized to attempt to prove that the number of resistors can go up to infinity. The experiment is a form of inquiry into the prospect of modelling circuits using computers, which has numerous potential applications. It will also attempt to speculate, depending on the results, if $P = NP$.

1 Introduction

Given that the experiment incorporates some higher-level concepts within Computer Science, it's appropriate to outline them here for a proper grasp of the experiment's goals/function. Please note, however, that the following is a low-level overview meant for a basic understanding. The final paper will approach Computer Science on a substantially more formal level.

1.1 Computational Complexity Theory

Computational Complexity Theory is a branch of Theoretical Computer Science that asks the question of *how hard* certain problems are (in terms of efficiency in solvency). There are two general classes of problems: the first consists of problems that *are* easily processed (by a computer), and has a set runtime of $O(n^k)$ (in other words, a static polynomial function). Problems here reside within the complexity class **P** (Polynomial Time), and are described as 'P-Complete.'[1]

For example, Consider a computer program that compares an input value with each number of an array of numbers.[1] It would be classified as $P - Complete$ as it has a set runtime of $O(n)$ (the program goes through all the numbers in the array). Almost all the software a typical person uses is composed of $P - Complete$ altorithims (e.g. Microsoft Office).

The other complexity class composes of problems that are *hard* for a computer to solve. There is no distinct $O(n^k$. Rather, a computer can *check* solutions to the problem in P-time. Problems of this class are said to be NP-Complete, or 'Non-Deterministic Polynomial Time.'

A good example of a NP-Complete problem is Minesweeper. This conceptually fits: a computer won't be able to exactly model problems based on *random chance* (such as the mines' locations); there will always be scenarios where the algorithim is wrong.[2] This further implies that computers can't model *everything.*

2 Appendix

Listing 1: Sequential Search

```
def sequentialSearch(test, arr):
        output = -1
        count = 0
```

[1] Code is available in the appendix

```
run = True
#the method ieterates through the function
#demonstrates complexity O(n)
while run and count < len(arr):
        if arr[count] == test:
                output = count
                run = False
        else:
                count+=1
return output
```

References

[1] Anil Maheshwari and Michiel Smid. *Introduction to Theory of Computation*. Carleton University, Ottawa, Canada, 2017.

[2] Richard Kaye. Minesweeper is np-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.