Some Problems are more Complete than Others.

Yash K. Sonune (002337133), ysonune22@ubishops.ca

Madhusudhan Jayaramu (002343190), mjayaramu22@ubishops.ca

Department of Computer Science, Bishop's University

CS590: Master's Project

Dr. Stefan D. Bruda

August 11, 2023

1. Abstract

This thesis focuses on the challenges of finding efficient solutions to computational problems in the dynamic landscape of computational complexity. It explores the potential of parallel computation as a promising approach to address the increasing demands for computational power. The research centers on P-complete problems, which are known as the most challenging problems. Investigating the feasibility of solving P-complete problems in parallel can provide valuable perception of boundaries and benefits of parallelism for complex problem-solving.

The significance of this research lies in its potential to uncover the limits of efficient computation and guide the development of more effective problem-solving techniques. By examining the relationship between P-completeness and parallel computation, the study aims to shed light on the advantages and limitations of parallel approaches.

The research questions address the feasibility of solving P-complete problems efficiently using parallel computation, the implications of parallelism for practical applications, and its impact on the boundaries of efficient computation. The relationship between parallel computation and POLYLOGSPACE will also be investigated. Combining knowledge from previous research and a comprehensive analysis of P-complete problems, the study aims to uncover their characteristics and assess the potential advantages and challenges of parallel computation.

2. Preliminaries

Here, we will establish the foundational concepts, notations, and complexity classes that will be utilized throughout the thesis. Understanding these preliminary notions is essential for comprehending the subsequent discussions on P-complete problems and parallel computation.

Decision Problems and Complexity Classes

It is a type of computational problem in which the goal is to determine whether a given input satisfies a specific property or condition. The possible answers to a decision problem are typically binary, either "yes" or "no". Our research thesis is confined toward Parallel Computing and Space Complexity (Log-Space). So, the crucial complexity class are s L, NL, P, POLYLOGSPACE, and P-Complete (Nondeterministic polynomial) time. NP contains decision problems for which a proposed solution can be verified in polynomial time.

Notations and Definitions

To facilitate precise communication and analysis within the field of computational complexity, specific notations and definitions are established. One common notation is the big-oh notation (O notation). In addition to the O notation, other notations such as omega (Ω) and theta (Θ) provide lower and tight bounds, respectively, on the growth rate of functions. We will also represent Deterministic Turing Machine as TM and Non-Deterministic Turing Machine as NTM. These notations allow for a more nuanced analysis of problem complexity.

Parallel Computation and Space Complexity

Parallel computation involves the simultaneous execution of multiple tasks or operations using multiple computational resources. Within the context of computational complexity, parallel computation introduces parallel complexity classes (NC). These classes consider the resources required by parallel machines, such

as the number of processors and the memory space utilized. The problem occurs when we try to relate Sequential Computation with Parallel Computation, as it is not directly possible. With Space Complexity classes (such as class is POLYLOGSPACE), which encompasses problems that can be solved in polylogarithmic space we can quantify the performance of parallel computation.

3. Tasks

3.1 Proof that NL ⊆ P and NL ⊆ POLYLOGSPACE (TASK I)

To show NL ⊆ P, we must give a logarithmic space simulation that runs in polynomial time, since problems for NL only run in Nondeterminism Logarithmic space machines. So, we would simulate that machine using a deterministic polynomial time machine. Let's say, we have a Non-Deterministic Turing Machine (NTM) named M that can decide a problem A using very little memory (space O(log n)). To analyze the behavior of M on input w, we create a "configuration graph" called $G_{(M,w)}$. The configuration graph $G_{(M,w)}$ has nodes representing all possible configurations of M while processing input w. A configuration includes the state of the machine, the position of the read/write head, and the contents of the tape. The graph has edges connecting configurations that can be reached in one step from another configuration. In other words, if M can move from configuration ci to configuration cj in a single step, there will be an edge from ci to cj in the graph.

Underlying, M will accept the input w if and only if there is a path in the configuration graph $G_{(M,w)}$ that starts from the "start" configuration (c_{start}) and reaches the "accept" configuration (c_{accept}). We can use this information to create a polynomial-time algorithm T for the problem A. This algorithm T would take an input w, and It would construct the configuration graph G(M,w) for the input w using the NTM M. Next, it checks if there is a path in the graph $G_{(M,w)}$ that starts from the "start" configuration (c_{start}) or reaches the "accept" configuration (c_{accept}). If such a path exists, it means that M accepts the input w, and Algorithm T accepts the input w as well. On the other hand, if there is no path from c_{start} to c_{accept}, it means that M does not accept the input w, and Algorithm T rejects the input w.

In conclusion, the algorithm T correctly decides whether the NTM M accepts the input w or not. Since we can construct this algorithm in polynomial time, therefore we have a deterministic polynomial time simulation of our NTM M. And it shows that problems that can be decided by an NTM using logarithmic space can also be solved in polynomial time using a deterministic algorithm.
Thus, NL ⊆ P.

Now, to prove (NL ⊆ POLYLOGSPACE) that NL (the class of problems decidable by NTM using logarithmic space) is contained within POLYLOGSPACE, Based on (IAN PARBERRY, 2, pg.53) the Savitch's Theorem,

$$NSPACE(S(n)) \subseteq DSPACE(S(n)^2)$$

However, Savitch's Theorem is not only confined to the sublinear or polynomial space. It can also be used for polynomial log-space.

$$NSPACE(S(\log n)) \subseteq DSPACE(S(\log^2 n))$$

This shows that a non-deterministic logarithmic space can be simulated by a deterministic space that is only larger by polynomials. Thus, we can say,

NL ⊆ POLY-LOGSPACE

3.2 Reductions (log-space) for Class P-complete (TASK II)

Reductions in complexity theory is a way of relating the difficulty of different computational problems, by showing that one problem can be transformed into another problem in a certain amount of time or space. If we can show that every problem in P is reducible to a problem C, and C belongs to P, then C is P-complete, meaning that it is the hardest problem in P. The concept of reduction is needed to show P-completeness. For P-complete problems, we need to use a log-space reduction, which means that we can change one problem into another using very little memory.

However, there is a catch while using log-space reduction. We cannot construct a Turing Machine (TM) which on input x computes f(x) in log space, and then simulates the poly-log space recognizer for B on its output. That would be an erroneous approach and works only perfectly well for polynomial-time reductions but fails for log-space reductions since it necessitates writing f(x) (which may be extremely large; since f(x) is computable by a log-space Turing machine, it may have size as large as a polynomial in n) on the work-tape. A better approach would be to compute the output f(x) bit-by-bit for each input x.

Let's say we have two languages or sets of strings called A and B. We want to understand if problem A is at least as hard as problem B. To do this, we use a function called f that takes a string from A and converts it into a string from B.

Now, we have some rules for this function f. The function f should not use too much space to do its work. It can only use a limited amount of memory, which grows slowly as the input size increases. We call this limited space "logarithmic space," represented by O(log |x|). The |x| represents the length of the input string x. The function f must be able to compute values in a way that is efficient and works for all inputs within a certain size limit. Specifically, the function f should be able to compute values for strings (x, i), where i is a number, such that f(x) has a 1 at the i-th position. Additionally, the function f should be able to handle values of i up to the length of f(x). Using this function f, we can compare the problems A and B. We say that A is logspace reducible to B (denoted $A \leq_{log} B$) if we can find a function f that meets the requirements mentioned above. For every string x in A, f(x) should be in B, and for every string x not in A, f(x) should not be in B.

Definition: logspace reduction
By the definition (S. ARORA and B. BARAK, 3, pg.84), we can think of a single O(log |x|)-space machine that given input (x, i) outputs $f(x)|_i$ provided $i \leq |f(x)|$. Language A is logspace reducible to language B, denoted $A \leq_{log} B$, if there is a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ that is implicitly logspace computable and $x \in A$ iff $f(x) \in B$ for every $x \in \{0, 1\}^*$.

Logspace reducibility satisfies usual properties one expects.

(a) If $A \leq_l B$ and $B \leq_l C$ then $A \leq_l C$.
(b) If $A \leq_l B$ and $B \in L$ then $A \in L$.

To prove this, let's considers two logspace implicitly computable functions, denoted as f and g, and their respective logspace machines Mf and Mg. Now we can Introduce a new function h(x) = g(f(x)) (for machine Mh), which represents the composition of these two functions. The idea is to show that h(x) can also be logspace implicitly computed. The machine Mh simulates the operation of Mg using a simulated fictitious input tape that holds f(x), while the actual input tape holds x. Mh maintains an index i that tracks the position of Mg on the fictitious tape and uses this information to guide the simulation of Mg's computation. Mh will simulate the computation of Mg up to the point where Mg would read $f(x)|_i$, the i-th bit of f(x). To get the

i-th bit of f(x), Mh temporarily suspends its simulation of Mg, invokes Mf on inputs x and i to get f(x)|i, and then resumes simulating Mg using this bit.

The total space used by Mh is O(log |g(f(x))| + log |f(x)| + log|x|), which simplifies to O(log |x|), considering that |g(f(x))| and |f(x)| are both bounded by |x|. In essence, the machine Mh simulates the behavior of the machine Mg while maintaining the necessary information about the position on the fictitious tape where f(x) is being read. And thus, this construction proves that logarithmic space reduction is closed under functional composition. That is, If $A \leq_{log} B$ and $B \leq_{log} C$ then $A \leq_{log} C$.

3.3 Relationship Between class P-Complete and POLYLOGSPACE (TASK III)

We can formally define P-complete problems as

Definition – A language L, is P-complete if,

1. $L \in P$, and
2. Every problem A in P is log space reducible to L.

Like the P and NP class relationship, the question that millennials are curious about is whether P and POLY LOGSPACE are the same class, meaning that every problem in P can be solved using very little memory (POLYLOGSPACE). But the answer is still not known. In light of the Parallel Computation Thesis, P-complete problems can be solved in polylogarithmic time in parallel if and only if every problem in P can be solved in polylogarithmic Space. If we can solve any P-complete problem using polynomial log-space memory, then we can solve every problem in P using very polynomial log-space memory. This would mean that P and POLYLOGSPACE are the same class. That is P = NC, when we prove P = POLYLOGSPACE. Every computation of a TM with space complexity S(n) can be simulated in a parallel computing model in time $T(n)=O(S(n)^{O(1)})$ and every computation of a parallel computing model with time complexity $T'(n)$ can be simulated by a TM with space complexity $S'(n)=O(T'(n)^{O(1)})$.

If a P-complete problem can be solved using POLYLOGSPACE, it implies that the problem's complexity is not higher than that of problems solvable with a polylogarithmic amount of memory space. In other words, this P-complete problem doesn't require more resources (specifically memory space) than what POLYLOGSPACE allows. Since the problems in P can be reduced to the P-complete problem, and the P-complete problem can be solved within POLYLOGSPACE, this implies that the entire class P can also be solved using a polylogarithmic amount of memory space. In other words, the problems in P do not require more memory space than what POLYLOGSPACE allows. Hence, showing that a P-complete problem is in POLYLOGSPACE implies that $P \subseteq$ POLYLOGSPACE. If we also assume that POLYLOGSPACE $\subseteq P$, which follows from the fact that any log-space transducer can be simulated by a polynomial-time machine, then we can further conclude that P = POLYLOGSPACE.

If P-complete problems are in POLYLOGSPACE, then this would imply that P = POLYLOGSPACE (as explained earlier. This would be a surprising result, since it would mean that all polynomial-time problems can be solved using polylogarithmic memory, and hence very efficiently. It would also imply that P = NC, since polylogarithmic space implies polylogarithmic time on a parallel computer. This would mean that all polynomial-time problems can be effectively parallelized, and there are no inherently sequential problems in P. Furthermore, it would imply that P = L, since logarithmic space is a lower bound for polylogarithmic space. This would mean that all polynomial-time problems can be solved using only logarithmic memory, and there are no super-logarithmic space problems in P. This would also mean that parallel computing can

solve any problem in P efficiently, which is widely believed to be false. Therefore, it is very unlikely that any P-complete problem is in POLYLOGSPACE.

On the other hand, If P-complete problems are not in POLYLOGSPACE, then this would imply that P ≠ POLYLOGSPACE, and hence there are some polynomial-time problems that require more than polylogarithmic space to solve. This would be consistent with the widely held belief that P ≠ NC and P ≠ L, and that there are some problems in P that are inherently sequential or require superlogarithmic space. However, this would not necessarily imply that POLYLOGSPACE is strictly contained in P, since it is possible that some problems in POLYLOGSPACE are not in P. Therefore, finding that P-complete problems are not in POLYLOGSPACE would leave open the possibility that POLYLOGSPACE contains some non-trivial problems that are neither in NC nor in L.

3.4 P-Complete Problems (Task IV)

P-complete problems are a class of decision problems that are both in P and hard for P. This means that they can be solved in polynomial time by a deterministic Turing machine, but they are also the most difficult problems in P, in the sense that any other problem in P can be reduced to them by an efficient transformation. P-complete problems are useful for studying the limitations of parallel computing and space-efficient algorithms, as well as the relationship between P and other complexity classes.

3.4.1 The First P-complete Problem – PATH

The notion of P-completeness appeared at roughly the same time as that of NP-completeness, but for a very different reason. The motivation was to study the relationship between sequential time and space. Cook raised the question of whether everything computable in polynomial time is also in polylogarithmic space. That is, the motivation was to ask whether a Turing machine running in time $n^{O(1)}$ could in general be simulated by one operating in space $(\log n)^{O(1)}$. Cook did not answer this question, but his 1973 conference showed that this question was equivalent to the question of whether a single, specific language, Path Systems, could be recognized in space $(\log n)^{O(1)}$. Cook showed PATH Systems (RAYMOND, 1, pg.111) to be complete for P, obtaining the first P-completeness result! Like all fundamental complete problems, the reduction Cook gave was a generic one for simulating a polynomial time-bounded Turing machine, using Path Systems in this specific case. The exact relationship between P and polylogarithmic space remains open, although we do know that polylogarithmic space cannot equal P; see Section 4.3 for more details. Since space complexity was the focus, Cook used a space-bounded form of reduction, logarithmic space reduction. It required that the algorithm accomplishing the reduction be restricted to using O(log n) space on inputs of length n.

3.4.2 The Second P-complete Problem – CVP

In 1975, Ladner showed that the Circuit Value Problem (CVP) was P-complete. Ladner's proof hinged on the observation that the transformation of the Turing machine into a circuit could be accomplished in logarithmic space. This means that any problem in P can be transformed into an instance of the circuit value problem using a polynomial-time algorithm that uses only constant-depth circuits with unbounded fan-in gates. Therefore, the circuit value problem is one of the hardest problems in P and solving it efficiently would imply that P = NC, which is widely believed to be false.

The circuit value problem is a decision problem that asks whether a given Boolean circuit evaluates to true or false on a given input. A Boolean circuit is a program that consists of finitely many assignments of the

form Pi := 0, Pi := 1, Pi := Pj Pk, j, k < i, ∧ Pi := Pj Pk, j, k < i, or ∨ Pi := Pj, j < i, ¬ where each Pi is a Boolean variable and the operators ∧, ∨ and ¬ represent logical AND, OR and NOT respectively. The input is a truth assignment to some of the variables, and the output is the value of a designated variable.

3.4.3 Proof of CVP is P-complete.

To show CVP is p-complete, consider a language L in the class P, which means there exists a polynomial-time Turing machine that decides it. This Turing machine M runs in polynomial time and accepts inputs that belong to L. For each input and time step of the Turing machine, a specific set of gates is defined. These gates simulate the behavior of the Turing machine at each step, tracking its state, head position, and symbol being scanned. Different types of gates are used to simulate various aspects of the Turing machine. For instance, Let's say gates D track the state and scanned symbol at each time step; gates Q indicate the current state; gates S track the scanned symbols; gates P indicate whether a specific cell contains a certain symbol; gates C determine if a cell contains a symbol; gates W track if a symbol was written to a cell during a step; gates O track if a symbol was written during a step; gates E indicate if a cell retained a symbol; gates U indicate whether the head is not scanning a cell; and gates H indicate whether the head is scanning a specific cell. The output of the constructed circuit is the value of the Q gate associated with the final state of the Turing machine after its execution. If the Turing machine accepts the input, the corresponding Q gate output will be 1; otherwise, it will be 0.

The constructed circuit simulates the behavior of the given Turing machine, and thus, the problem of determining whether the output is 1 corresponds to the problem of deciding whether the input belongs to language L. Since the reduction from any language L in P to CVP can be done in logarithmic space (a space-efficient manner), CVP is in P. Moreover, any language L in P can be reduced to CVP. This means CVP is P-complete. A thorough formal proof can be found in the book referenced (Ref. IAN PARBERRY, 2, pg.58-60) "*Parallel Complexity Theory*".

3.4.4 Other P-Complete Problems

Aside from Path System and CVP, there are several other interesting P-complete problems that have been discovered over the years. Here are a few examples:

- Restricted Case of CVP: Based on different setups of binary circuit C, such as Monotone-CVP (C consists of AND and OR gates), NOR-CVP (C consists of NOR gates with fan-out 2), Planer-CVP (C is a planer Boolean circuit). (Ref. IAN PARBERRY, 2, pg.58-62)

- Linear Inequalities Problem (LI): Give an integer n × d matrix A and an integer n × 1 vector b, is there a rational d × 1 vector y > 0 such that Ay ≤ b? (where y > 0 mean all components of x are nonnegative and at least one is nonzero). (Ref. RAYMOND, 1, pg.150)

- Linear Programming Problem (LP): Linear Programming is a method for optimizing a linear objective function subject to linear equality and inequality constraints, to find if a feasible solution exits for problems (such as described earlier in LI problem).

- Lexicographically First Depth First Search Ordering – Given a graph with fixed ordered adjacency lists, and nodes u and v, is vertex u visited before vertex v in a depth-first search induced by the order of the adjacency lists?

- Context Free Grammar Membership – Given A context-free grammar G= (N, T, P, S) and a string x, can that string be generated by that grammar?

4. Methodology, Results, Findings, and Discussion

This section encapsulates the research methodology, results, findings, and their implications. The primary goal was to investigate the feasibility of employing parallel computation for P-complete problems, the most challenging problems within the P complexity class. The research encompassed a thorough literature review, algorithm analysis, and complexity proofs to establish the inherent difficulty of P-complete problems.

The investigation into parallel computation involved exploring the benefits and limitations of parallel approaches for solving P-complete problems. Efficiency gains were found to depend on the complexity class POLYLOGSPACE. If a P-complete problem can be efficiently solved using parallel machines within polylogarithmic time and space, it suggests P equals POLYLOGSPACE. However, problems exceeding these bounds might not see significant advantages from parallel computation.

The findings emphasize the importance of understanding the characteristics of P-complete problems before choosing computational paradigms. While some problems are amenable to parallel solutions, others retain their sequential nature and may require alternative approaches like approximation algorithms.

The practical implications of the research provide valuable insights for algorithm designers and practitioners. Identifying P-complete problems aids in recognizing challenging tasks that may require specialized techniques, guiding the selection of appropriate computational methods. By exploring the interplay between P and parallelism, this research advances efficient problem-solving strategies and informs future algorithm development.

5. Conclusion and Future Work

This thesis has explored P-complete problems and their relationship with parallel computation. Through foundational works, reduction techniques, and complexity analysis, the inherent complexity and computational boundaries of P-complete problems are clarified. The findings underscore the challenges and potential benefits of parallel computation for certain P-complete problems.

Future research can expand on these findings by developing efficient approximation algorithms and exploring alternative parallel computation models like distributed computing or GPU-based parallelism. Investigating the boundaries of parallel computation within specific subclasses of P-complete problems can yield insights into practical applicability.

Furthermore, research can focus on creating parallelization frameworks tailored for P-complete problems, providing high-level abstractions and optimized implementations. These frameworks can empower researchers and practitioners to effectively harness parallelism for addressing real-world instances of P-complete problems.

In summary, this thesis lays the foundation for understanding P-complete problems and the potential of parallel computation. Future research should explore efficient approximation algorithms, alternative parallel models, investigate restricted settings, and create specialized parallelization frameworks. By

pushing the boundaries of efficient computation, researchers can advance the field of computational complexity.

6. References

1. RAYMOND GREENLAW, JAMES HOOVER, WALTER L. RUZZO, *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press, New York, NY, 1995, Ch. 4-6.

2. IAN PARBERRY, *Parallel Complexity Theory*, John Wiley & Sons, New York, NY, 1987, pg.50-68.

3. S. ARORA and B. BARAK, *Computational Complexity: A Modern Approach*, Princeton University, Cambridge University Press, 2009, pg. 84-86, https://theory.cs.princeton.edu/complexity/book.pdf

4. RICHARD E. LADNER, *The Circuit Value Problem is Log Space Complete for P* *, University of Washington, SIGACT News, 1975, *https://dl.acm.org/doi/pdf/10.1145/990518.990519*

5. MICHAEL SIPSER, *Introduction to the Theory of Computation*, Cengage Learning, 2013