Theory of Computation Final Research Paper

Space Complexity Applied to Game Theory

Zachary M. Shaffer

Allegheny College

Dr. Oliver Bonham-Carter

December 12, 2016

Computational complexity is a very broad topic that covers a lot of different areas in computer science and computational theory. One way we observe computational complexity is through the lens of space complexity. Moreover, when one studies space complexity, a wide variety of connections become available for research. In particular, while studying space complexity myself, I discovered that there are deep roots connecting the PSPACE classification of problems to game theory. When one studies space complexity, specifically the FORMULA-GAME problem in the PSPACE class of problems, the possibilities of finding optimal play algorithms for games like chess or checkers become both quantifiable and conceivable.

Computational complexity is a broad term that describes the study of computational difficulty and efficiency. Two of the most important subcategories of computational complexity in the study of computational theory are time complexity and space complexity. In the theory of computation, we tend to use Turing Machines to test the time and space complexity because they are both mathematically simple and close to real computers in their behavior (Sipser 331). Time complexity measures the number of steps the Turing Machine must take for any given input of a certain length. This effectively measures the time complexity of any given machine or algorithm because each computer takes a set amount of time to process a cycle, which can map one-for-one to the steps a Turing Machine must take on any given length of tape. Space complexity, however, is measured specifically by the number of tape cells any given input tape of a certain length. Rather than measure the number of steps, which take a set amount of time, we use the Turing Machines to measure the number of cells, which represent the amount of memory space taken up by any given machine or algorithm. Another very important piece of information is that space is generally more powerful than time. That is, space can be reused where time cannot be. This fact will become important later when discussing the specific classifications of problems in computational complexity.

The metric by which we discuss most forms of complexity is described by asymptotic notation. Asymptotic notation is simply a function g(x) such that the function f(x) that describes the number of steps or cells is less than or equal to the function g(x). We denote this using “Big-O”notation, or O(f(x)), for a function whose complexity is quantifiable by the function f(x). The function f(x) can be linear, exponential, logarithmic, cubic, or any other descriptor that describes a function. With that established, we can now observe different classes of problems. Earlier in the course, we discussed the P and NP classes of problems. P class problems are problems that can be quickly solved by a computer, and NP class problems are problems whose solutions can be quickly verified by a computer. “Quickly,” in the context of both of those problems, means that the problems can be reduced to polynomial time. This is important to establish because we describe particularly difficult problems in the NP class as “NP-complete.” Once an NP-complete problem is solved, however, we can reduce other problems with polynomial time solutions to that NP-complete problem. The reason this is all important to space complexity is because space complexity has a similar structure to it’s classes of problems. SPACE problems are problems that can be decided on a deterministic Turing Machine in linear complexity, O(f(n)), where f(n) is a linear function. NSPACE problems are problems that can be decided on a nondeterministic Turing machine in linear complexity (Sipser, 332). We mentioned before that space is more powerful than time, simply for the fact that space is reusable where time is not. This leads us to a powerful theorem by Walter Savitch, commonly known as Savitch’s Theorem.

Savitch’s Theorem, by definition, states that for any function f(n) where f(n) is greater than or equal to n, NSPACE(f(n)) = SPACE(f 2 (n)). In more colloquial terms, Savitch’s Theorem demonstrates that all space problems that are simulatable on a nondeterministic Turing machine are efficiently simulatable on a deterministic Turing machine, as well (Sipser, 334). This theorem is important because it introduces a unique relationship between two fundamental classes of problems in space complexity: PSPACE and NPSPACE. PSPACE is the class of problems that are decidable in polynomial space for

deterministic Turing machines, and NPSPACE is the class of problems that are decidable in polynomial space for nondeterministic Turing machines. Since we know that space problems that are simulatable on a deterministic Turing machine are likewise decidable on a nondeterministic Turing machine, thanks to Savitch's theorem, we know that problems in NPSPACE are also problems in PSPACE, thus PSPACE = NPSPACE. This relationship is unique to classes of problems, as it is widely believed that P problems are not the same as NP problems, but no proof has been established either way. Furthermore, we can assume that the relation of all complexity classes are as follows:

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME,$$

where EXPTIME is the classification of problems who are solvable in exponential time. None of these containment statements, however, have been or can yet be proven proper or improper. The best we can surmise is that one of the above containments is proper, by merit of the existence of a proof that states EXPTIME problems and P problems are not the same (Sipse, 336). Now that we have established the relationships and existences of the fundamental classes of problems, or at least the ones that will matter for examining game theory, we can proceed by accepting that PSPACE-complete problems, like NP-complete problems, are problems that are very difficult to solve and yet both useful and powerful when applying computational theory to real practices.

PSPACE-completeness is defined in a nearly identical way to NP-complete problems; but there is a catch to the definition. With NP-complete problems, we are looking for problems reducible in polynomial time to other problems. Rather than looking for the same in PSPACE-complete problems with polynomial space, PSPACE-complete problems actually also reduce in polynomial time as well. The reason for that is obvious: we define and search for NP-complete and PSPACE-complete problems so that when we reduce a problem down to an NP-complete or a PSPACE-complete problem, we can find a solution for the original problem. By reducing PSPACE-complete problems in polynomial time, we simplify problems to subspace levels, reducing them to mere time complexity problems. This

allows us to further reduce problems, bringing them closer to problems we can more quickly solve or verify solutions for. When examining game theory, PSPACE-complete and PSPACE-hard problems are what we work towards producing and solving.

With PSPACE completeness in mind, let's consider our first example of PSPACE applications in game theory: the formula game. The formula game is a game based on a fully quantified boolean formula. A boolean formula is any formula whose language is {0, 1} and uses boolean operators (AND, OR, and NOT) to determine a true or false outcome, where 0's indicate false and 1's indicate true. For the boolean formula to be quantified means that the variables used are defined by one of the two major quantifiers: the existential quantifier and the universal quantifier. The existential quantifier denotes the existence of a variable where a variable is used once, and the universal quantifier denotes when all of a variable share the same properties. Lastly, for a boolean formula to be fully quantified simply means that all variable in the scope of the boolean formula have a related quantifier. The problem of determining whether or not a true fully quantified boolean formula returns a true or false value is known as the TQBF problem, and it is proven to be PSPACE-complete (Sipser, 339). The formula game is a game where two players attempt to make a statement either true or false by taking turns determining values in a quantified boolean formula. And, as with most simple turn-based games, this game has a winning strategy. A winning strategy means that if both players play "perfectly," or optimally for all of their given turns, one of the players will win no matter what. The problem of determining a winning strategy in a formula game is known as the FORMULA-GAME problem. For similar reasons that we know the TQBF problem to be PSPACE-complete, we know the FORMULA GAME problem to likewise be PSPACE complete. This comes in handy for use when attempting to reduce the problem of finding a winning strategy for other games that we can likewise reduce to PSPACE problems.

By reducing problems to PSPACE-complete or PSPACE-hard problems, we can attempt to find a winning strategy for the games. Michael Sipser produces evidence that games such as the Geography Game can be reduced to PSPACE-complete or PSPACE-hard problems (Sipser, 345). The Geography Game is a simple game where players take turns naming countries. The first player picks a random letter, and the second player must name a country starting with that letter. The players then exchange turns naming countries whose starting letter is the same as the last letter of the previous player's word. This is the part of the research that really piqued my interest, as Sipser's demonstration that the Geography Game, when generalized to a nearly arbitrary game based on some graph, can be shown to be a PSPACE-hard problem and thus we can see that there is no optimal algorithm that can be made for playing the Geography Game. While this may sound like bad news, the implications otherwise are huge; other games, like chess or checkers, can be generalized and examined to determine their PSPACE-hardness or PSPACE-completeness. Furthermore, as we learn more about machines and the theory of the relationships between various complexity classes, we could potentially find palatable algorithms to optimize the play of certain games, depending on how exactly one generalizes the game. This shows great potential to help us design, produce, and test games to determine their PSPACE-hardness, and thus determine their effectual difficulty.

Overall, my research has opened up a whole new field of possibilities to me, as I had never before considered the relationship between complexity theory and game theory. My perfunctory research also drew my attention to a few other details about complexity theory that were less related to the main topic of my research, such as the difficulty in determining the specifics of the containments of the complexity classes with respect to one another. Regardless, while focusing my attention on space complexity, the possibilities and potential for this field of study, when applied to game theory, have become vastly more apparent in my mind's eye, and my research has left me with a piqued interest and a sense of excitement for the future of theoretical computational research.