The Role of the Goal in Problem Solving Hard Computational Problems: Do People Really Optimize?

by

Sarah Elizabeth Carruthers
B.Sc., University of Victoria, 2004
M.Sc., University of Victoria, 2008

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Interdisciplinary Studies

The Role of the Goal in Problem Solving Hard Computational Problems:
Do People Really Optimize?

by

Sarah Elizabeth Carruthers
B.Sc., University of Victoria, 2004
M.Sc., University of Victoria, 2008

Supervisory Committee

Dr. Ulrike Stege, co-Supervisor
(Department of Computer Science)

Dr. Michael Masson, co-Supervisor
(Department of Psychology)

Dr. Margaret-Anne Storey, Departmental Member
(Department of Computer Science)

Dr. Anthony Marley, Departmental Member
(Department of Psychology)

Supervisory Committee

Dr. Ulrike Stege, co-Supervisor
(Department of Computer Science)

Dr. Michael Masson, co-Supervisor
(Department of Psychology)

Dr. Margaret-Anne Storey, Departmental Member
(Department of Computer Science)

Dr. Anthony Marley, Departmental Member
(Department of Psychology)

ABSTRACT

Understanding how humans cope with complexity is perhaps one of the most important targets of scientific research. Humans not only excel at solving complex tasks in their day to day life but also take on objectively difficult problems recreationally. Research, which has focused to date on the famous hard optimization problem the Euclidean Traveling Salesperson problem (E-TSP), has indicated that humans are able to find near optimal solutions in linear time to hard optimization problems despite the objective difficulty of the task. The research presented in this work contributes to this research by comparing human performance on the search and optimization versions of two other visually presented computationally hard problems: Vertex Cover and Independent Set. These two problems were selected in part to explore how human performance might differ on not-Euclidean problems. Performance on the optimization version of the problems used in this study is in keeping the previously reported results; however, performance on the search version is even better suggesting that previous problem solving research might have

underestimated the power of the human problem solving system. A key result of this work is that differences in performance between the optimization and search versions of these hard problems can be attributed to differences in how problem solvers encode the goal of the task. Consequentially, subjects in these conditions tasked with identical instances of two very nearly identical versions of a problem are in fact solving very different problems. This work presents a framework to improve how human performance results on hard optimization problems are interpreted. It also demonstrates how the search version of hard computational problems can be used to investigate how people cope with complexity free of the confounding aspect of an ill-defined goal.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

The Community of Lasqueti Island, for keeping me inspired.

Ulrike Stege, for endless support, and patience.

NSERC, for the Fellowship that made this work possible.

Tennyson said that if we could understand a single flower we would know who we are and what the world is. Perhaps he meant that there is no deed, however so humble, which does not implicate universal history and the infinite concatenation of causes and effects. Perhaps he meant that the visible world is implicit, in its entirety, in each manifestation, just as, in the same way, will, according to Schopenhauer, is implicit, in its entirety, in each individual.

Jorge Luis Borges - Labyrinths: Selected Stories and Other Writings

DEDICATION

This work is dedicated to my whole family, who have always lovingly supported my pursuit of knowledge:
Katherine Maas & Roberta Claire,
Jeff Carruthers & Mary Ferguson,
John Manson & Maggie McGovern,
Rebecca & Don Carruthers den Hoed,
Bryne Carruthers,
Mal & Zach Carruthers den Hoed,
Cedar Cove, and Kaleb Cove Dias.

Chapter 1

Introduction

Like many computer scientists, I find great joy in the challenge of problem solving, be it in the form of games or puzzles, or discovering and solving a new hard problem. It is inherently interesting to me *why* people enjoy solving problems, an interest that directly motivated this work. In computer science, the study of computational complexity has many tools and techniques for classifying problems based on their objective difficulty, as well as techniques for coping with problems that resist efficient solution. These tools and techniques were adopted by cognitive scientists to try to better understand the power of the human cognitive system. It became possible to think about the cognitive system, and more specifically problem solving, from a formal computational perspective. Computational complexity also brought some very hard problems into the limelight, eventually leading to their adoption as instruments to investigate human problem solving.

The results of these studies have been somewhat surprising, showing that people are quite adept at finding close to optimal solutions remarkably quickly, seemingly at odds with what is thought to be known about these problems. It was these findings that planted the seed that would become this work. It began with a desire to see if the results of earlier studies were representative of problem-solving proficiency on visually presented hard problems, in general, but evolved into an investigation of how people cope with a specific facet of the challenge of these problems. That is, how does the human cognitive system cope with a problem-solving task environment in which it is not necessarily possible to know when a solution is correct? This kind of challenge is one that is quite natural. Humans have evolved in a world riddled with tasks for which there is no meaningful way to determine when the best answer has been achieved. Therefore it is likely that we have also developed mechanisms for

quickly coping with this kind of complexity.

The reader will note that this introduction is brief. Due to the interdisciplinary nature of this work, a great deal of introductory material is presented in Chapter 2, as two very different perspectives of problem-solving theory must be addressed. Rather than try to compress this into a traditional introduction, this introductory information is woven into the description of the nature of the problem tackled by this research. The hope is to situate the introductory material in a way that is meaningful and approachable to readers from both disciplines: cognitive science and computational complexity.

1.1 Overview

Chapter 2 begins with an overview of problem-solving theory, first from the cognitive science perspective, followed by an introduction to computational complexity. Relevant previous work studying human performance on hard optimization problems is presented, and a previously unrecognized shortcoming of using optimization problems as instruments to study human problem solving is identified.

Once the necessary groundwork has been laid, Chapter 3 presents two methods to remedy the identified shortcomings of previous work. The first method suggests how previous study results might be re-interpreted. This includes a framework for decomposing the goal of hard optimization problems to facilitate a better understanding of problem-solving performance on these problems. As an alternative, another problem formulation is presented to allow for the study of human problem solving on hard problems that are devoid of the identified shortcomings.

Chapters 4 and 5 present the results and analysis of this research study. The framework identified in Chapter 3 is used as a template to demonstrate how to interpret performance results on hard optimization problems in light of the shortcoming identified in Chapter 2. This study also uses the other problem formulation identified in Chapter 3, as an alternative to hard optimization problems, allowing for comparison of relative performance between the two versions of each problem. Chapter 5 presents a detailed analysis of human performance on both versions of both problems used in this study.

A computational model is presented in Chapter 6, designed to test the theory that simple local strategies might be sufficient to explain individual human performance on the Vertex Cover problem used in this study. This model was key in identifying

interesting problem-solving behaviour, which then informed the overall analysis.

Chapter 7 presents a summary the findings of the study and their implications, along with a number of questions suggesting future areas of research to further extend our understanding of how humans cope with complexity, and the power of the human cognitive system.

Chapter 2

Coping with Complexity

> "There are no more promising or important targets for basic scientific research than understanding how human minds, with and without the help of computers, solve problems and make decisions effectively, and improving our problem-solving and decision making capabilities." [96], p. 12.

The main drive of this work is to contribute to the study of how people cope with complexity, motivated by a number of observations, including: the prevalence of hard problems in day-to-day life; the relative ease with which people are able to handle hard problems; and people's attraction to complexity for diversion and entertainment. This work will explore in greater precision in the following sections what can make a problem *hard*, or *difficult*, but for now this work will consider problems that are composed of many variables, and those problems which are subjectively difficult.

Day-to-day life is rife with hard problems. Getting to work or school can start with trying to find a fastest route; at work, managers try to efficiently schedule workers and resources, while workers might be trying to maximize the number of customers who are satisfied by the company's service; after work, shoppers insistent on a single trip from the car need to figure out how to fit their groceries into a limited number of sacks. These problems do not necessarily seem subjectively hard to many people, and are typically completed without too much difficulty. But, as will be shown in Section 2.2, many of these tasks are hard in an objective sense. The human cognitive system's ability to tackle complex problems like these with what can sometimes seem like little or no effort is fascinating and deserving of continued study.

The fact that people deal with hard problems in day to day life is perhaps un-

avoidable. The world is complex, and maybe, given the choice, people would avoid activities or tasks which are difficult if they could. However, the opposite appears true. People are often attracted to hard problems for recreational purposes, and in fact, many popular games and puzzles have been shown to be objectively difficult (see Section 2.2 for details). Popular video games which have been proven to be difficult include: Tetris, Minesweeper, Mastermind, Mario, Donkey Kong, Legend of Zelda and Pokemon [2, 22, 88, 100]. Puzzles and analog games that have been shown to be hard include: 15-puzzle, Battleships, Rush Hour, and a number of wooden geometric puzzles [3, 42, 81, 89]. It is unknown what attracts people to hard games and puzzles, but whatever the explanation, the attraction exists despite the fact that these problems have defied efforts to find general, fast, solutions. This fact has not gone unnoticed, and games have been used to make it easier to teach hard concepts since the advent of digital games. Recent work in this area has used games and simulation to teach complex topics, including business management and problem solving [50, 75, 99]. Better bridging of the gap between the objective and subjective measures of difficulty can serve to inform this effort. Better understanding what aspect of these hard games makes them addictive and engaging has the potential to improve educational games.

Gaining an understanding the power of the human cognitive system and why people do not balk at confronting hard problems also has the potential to inform education, workplace practices, and interface design. Problem solving skills are key to many levels of study, from kindergarten to graduate school, and many areas of study, including but not limited to mathematics, sciences, and engineering. Similarly, this kind of study can lead to a better understanding of what properties of problems or the human cognitive system make problems either engaging or boring. This knowledge could lead to better methods for supporting problem-solving skill acquisition, by identifying problems which are difficult yet engaging. As suggested by Polya in [78], stimulating interest in the joys of problem solving is key to building strong problem-solving skills, and could serve to inform educational practices. Similar claims can be made about work environments, which can include complex problems that must be solved. Scheduling conflicting tasks or resources has been shown to be a hard problem, but it is one which many managers face on a regular basis. Studying human performance on hard problems can help to gain a better understanding of what properties of problems contribute to their difficulty, and what aspects of the cognitive system contribute to the perception of the subjective difficulty of prob-

lems, making some problems seem subjectively easy even if they might be objectively hard. A better understanding of what cognitive processes can be leveraged to make complex or difficult tasks easier, could improve the ease and efficiency of common workplace tasks. It is well established that problem representation impacts performance on tasks [69], and it follows that understanding how problem-solving tasks are represented must also impact interaction with tools and technology used to process or solve those problem-solving tasks. Work in this area has already made advances, by determining that problem and information representation can impact problem-solving performance [12, 60, 71].

Problems and problem solving occur in many parts of our daily lives: at school, at work, at play, and in our daily routines. Better understanding how humans cope with complex problems can give insight into how problem solvers manage to deal with many kinds of problem, and also help design education, workplaces, games, and tools to leverage the power of the human cognitive system.

2.1 Problems, Problem Solving, and Difficulty

The terms *problem* and *problem solving* may have different meanings depending on context, and therefore it is necessary to more carefully define what is meant in this context by these terms. This is especially important as this work is interdisciplinary in nature and terms often differ in meaning and usage between disciplines. Computer science and psychology are no exception to this pattern. First, the psychology perspective of problem solving will be explored, presenting typical definitions of problem solving, eventually narrowing down to the definition of interest here. There are many kinds of task that could be considered suitable for studying problem solving [34], and various ways of classifying the difficulty of such tasks. What is meant by *problem*, *difficulty*, and *problem solving* in the context of this work will be defined more precisely. In particular a class of problem is identified that can be clearly and exactly presented to problems solvers such that the problems contain all information necessary to solve them, and are free of uncertainty and unknowns. Once the kind of problem-solving task that is of interest has been sufficiently defined, the focus will turn to problem solving in the context of computer science, to find common ground between these two fields. The goal is to strengthen the foundation of empirical problem-solving research by building upon previous work in these two fields.

2.1.1 What is Meant by Problem Solving?

In the field of psychology, a task is typically considered to be a problem if there are obstacles that make achieving the goal a challenge [66], and problem solving is the process of determining a way of overcoming the obstacle.

The first distinction that must be made is the difference between *problem solving*, and *decision making*. Problem-solving research in many ways has its roots in decision-making research [91, 92, 93, 94]. Decision making takes place when people are required to, very quickly or under pressure, make *good* decisions under conditions of uncertainty, unknowns and variability, e.g. choosing a best investment in the stock market. Even the most experienced and skilled individual cannot possibly know all the factors that contribute to a stock's success or failure, and there is no definitive way to evaluate what constitutes a *good* solution. Regardless, decisions like this are made all the time, often (at least by subjective measures) quite successfully. While decision-making theory is an important area of research, it is not of interest here because the goal is to limit the scope of this work to problems that can be presented free of unknowns and uncertainty.

2.1.2 Examples of Problem Solving Tasks

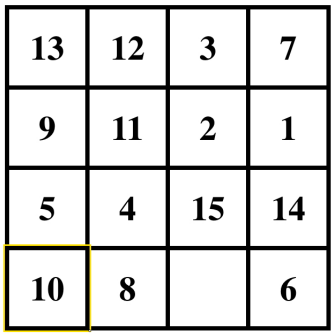
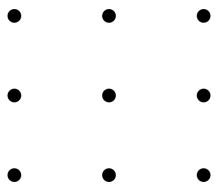
Defined as a task for which an obstacle impedes the path to the goal, many games and puzzles are examples of problem-solving tasks, for instance 15-puzzle, 9-dot-puzzle, and chess, shown in Table 2.1. These three problems have been used in problem-solving studies, and serve as a good introduction to some of the properties of problems that are of interest in this work.

The 15-puzzle is a simple, popular game that consists of numbered tiles within a constrained square, and the task is to slide the tiles until they are rearranged in numerical order. The challenge, or obstacle, of this puzzle is rooted in its combinatorial complexity, in the number of possible sequences of moves that might lead to the goal.

The 9-dot-puzzle is a very well-known problem in the study of problem solving. Given nine dots in a square as shown in Table 2.1, the goal is to connect all nine dots by drawing four straight lines without lifting the pen/pencil from the paper. This is an example of an insight problem, one that many people struggle to solve. Though seemingly simple, the obstacle of this puzzle is often found to be in the correct interpretation of what constitutes a legal move.

Chess is a two player game that is very well known for its difficulty, and can take

Table 2.1: Three puzzles/games used as instruments in problem-solving studies.

Problem Name	
15-puzzle	
9-dot-puzzle	
Chess	

many years to master. The challenge or obstacle of this problem stems from the number of possibilities that must be considered when trying to find a best next move. A chess player must, at each of their turns, try to determine a move that leads to a state where they are the winner.

All three of these examples constitute problem-solving tasks, and have each been the subject of human performance studies [57, 66, 76]. Now consider, informally, how these problems differ in terms of whether they can be presented without unknowns or uncertainty. The 15-puzzle problem can be clearly defined in terms of input, legal moves, and goal. Moves are clearly defined: a tile can be slid into an adjacent empty slot, if one exists. Depending on the location of the empty slot, this means that between two and four tiles are moveable at any given state. The goal of the puzzle is also clearly defined: order the tiles. As such, it lacks the uncertainty, unknowns, and variability associated with decision-making tasks. The 9-dot-puzzle has a simple well-defined input and goal, and seemingly well-defined legal operators. However, the way the problem is presented or interpreted often results in an ambiguity that makes achieving the goal very difficult, if not impossible, to achieve. The reasons for this have been studied extensively [6, 15, 18, 21, 43, 57], and have been attributed to assumptions about what constitutes a legal move that could be a result of previous experience with points in the plane, or some other aspect of the visual stimulus. Whatever the reason, the legal moves are ambiguous, and as a result the problem is rendered ill-defined for many problem solvers. Finally, chess can be clearly described such that it lacks uncertainty, unknowns, and variability. The start state is simple and well-defined, the legal moves are equally well specified for each piece, and the goal state is simple: checkmate the King. It differs from the 15-puzzle in that it is a two-player game, rather than a one-player game. Newell and Simon [66] used chess as an instrument to evaluate problem solving by giving participants an intermediate state of play, and asked them to determine the best possible next move, rendering it possible to present this typically two-player game as a one-player problem-solving task.

15-puzzle and chess both possess desirable qualities: clearly defined start states and goals, a clearly defined set of legal operators. Both of these problems can be presented such that they are suitable for single-player environments, and can be classified as problems of transformation [34]. Solving them amounts to finding a sequence of moves or operations to transform the start state into a goal state. Transformation problems also include such well-known problems such as: Towers of Hanoi, Water-

Jug problems, and math word problems, many of which have been used extensively in human performance studies [8, 7, 49, 17, 71, 101].

2.1.3 Problems of Transformation

More formally, a transformation problem consists of a well-defined start state, a well-defined and finite set of legal operators, and a well-defined goal state. In order to be well-defined the state or operator must be encodable and it must be possible for it to be evaluated by the human cognitive system. As such, it not only must be finite, but also constrained to the limits of the human cognitive system, including but not limited to perception, working memory, and processing. The limits of the human system can be extended to some extent through the use of external aids, such as interactive representations of the current game or puzzle state. This is common in games like chess, where players typically rely upon the physical representation of the game to recall the current state of the board.

Transformation problems provide a good starting point for studying human problem solving under controlled conditions because they are a class of problems that are devoid of uncertainties and unknowns. This restriction to transformation problems eliminates a number of kinds of problem that have previously been considered in studies of human problem solving, notably decision-making and insight problems (See, e.g., [15, 18, 30, 43, 57, 91]). Studying decision making and insight can teach much about how humans cope with complexity in a world filled with unknowns and uncertainty. However, restricting study to transformation problems with well-defined start states, goal states, and legal operators has the advantage of allowing certain assumptions to be made about the internal representation used by the problem solver, which will be discussed in detail in the upcoming sections.

2.1.4 Information Processing Model of Problem Solving

Problem solving research as it is known today found its beginning in the late 19th Century with Thorndike who, driven by the belief that human consciousness evolved from the "processes of association" of lower animals, strove to determine the origin of human problem-solving ability by investigating problem solving of cats, dogs and chicks [108]. In this foundational work, Thorndike critically reflected upon previous work studying animal abilities, and made a conscious decision to improve upon its flaws, that is, to devise sound experiments to test theories, rather than rely upon

anecdotal evidence. This was a fundamental shift, upon which modern problem-solving research was built.

Maier built upon this foundation, and his attempts to unravel the relationship between experience and reasoning using problems of transformation [58], led to the development of problems that were progressively more controlled and well-defined. The adoption of these controlled instruments in human problem-solving studies made it possible to empirically determine, for example, that experience with instances of problems could impact performance on subsequent instances, sometimes in surprising ways. Luchins' concept of a solution to a problem being a sequence of moves [51], and the emergence of the concept of productive thinking [27, 59] laid the groundwork that made it possible for problem-solving research and computation to find common ground.

The advent of powerful computers in the 1950s was an important factor in the rise in popularity of computational theories of mind, and is a great example of how the emergence of new tools can influence the advance of scientific theories [31]. Being able to think about problem solving from a computational perspective allowed for a wealth of knowledge about problems and problem solving from the field of computer science to be transferred to the study of human problem solving. It also made implementing working models based on these theories much more feasible. Problems of transformation were ideally suited for information-processing models of human problem solving, like those proposed by Newell and Simon in the late 1950s [67] and later refined in the 1970s [66]. Their information-processing model of human cognition, and in particular problem solving, motivated a great number of experimental studies using problems of transformation. This approach also laid the groundwork for the advancement of artificial intelligence [47, 95].

One of the fundamental concepts that came out of the information processing model of problem solving is that of a *problem space*. When a problem solver is tasked with a problem, they generate an internal representation, or problem space [66], that can be thought of as a representation of all possible paths from the start state, through all reachable states, to all goal states. The transition from one state to another in the problem space is determined by legal operators that can be applied at a given state. Problems of transformation are thus named because each move transforms the state of the problem into a new state, until eventually a goal state is achieved. This problem space exists in the sense that the entirety of the problem space is reachable via legal operators from any current state, even though the problem solver does not

necessarily generate or retain the entire problem space in memory (either short or long term). At any given moment, the problem solver may be only locally aware of the problem space, the current state, and those states that are reachable through the application (or undoing) of a small number of operations, as well as perhaps some number of previously visited states. It is useful to think about the problem space in its entirety even though it rarely exists in that pure form, in particular when trying to investigate what might make a problem objectively or subjectively difficult.

As a example, consider the problem space of the problem used in [7]. In this study, participants were given instances of the 3-Water-Jug problem. In this problem the start state consists of three jugs, (A, B, C) a capacity for each jug, and a starting fill-level of water for each jug. The goal state consists of a given fill-level of water for all jugs. The subject is asked to find a sequence of legal pouring steps that transforms the start state to the goal. A legal pouring step involves transferring water from one jug to another, such that the overall volume of water in the three jugs remains constant. The instance of the problem used in this study, consisted of three jugs (of size 8, 5, 3), a start state of the eight-unit jug filled, and a goal state of four units each in the eight-unit and five-unit jugs. The problem space in Figure 2.1 from [7] corresponds to this instance of the 3-Water-Jug problem. Lines in the problem space represent legal pouring steps, and the single goal state is at the bottom of the graph. There are two direct paths to the single goal state for this instance of the problem. This problem space includes loops (denoted by links to nodes labelled 'R', 'S', 'T' and 'L') because it is possible through the application of some sequences of moves to return to some states. Problem solving of this task can be thought of as the process of finding a sequence of moves that defines a path from the start state to the goal state.

So long as the given legal operators are sufficient to ensure that there exists at least one path from the start to the goal, then it can be assumed that the problem solver's problem space also contains a path from the start to the goal, and therefore the problem is solvable. It is assumed hereafter that when referring to problems of transformation in studies of human problem solving, the problem set is restricted to those instances for which a solution is achievable given the problem's definition.

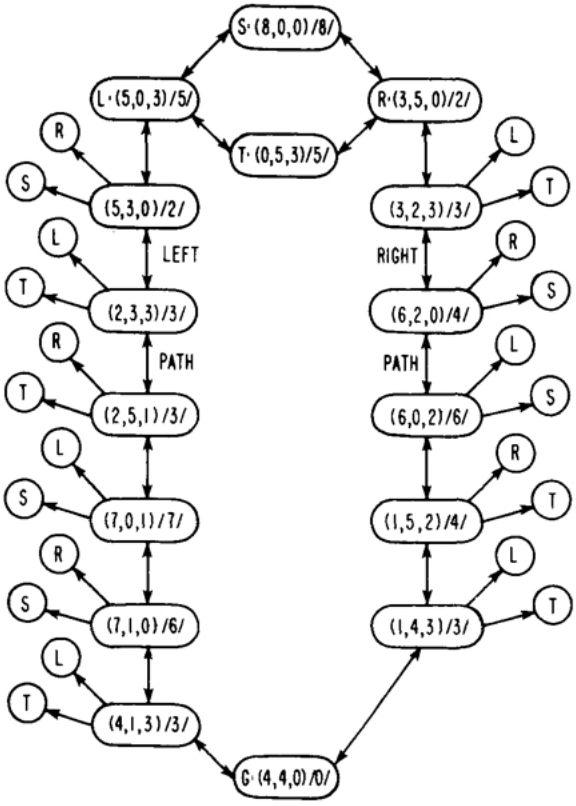


Figure 1. Graph of the possible states and legal moves for the (8,5,3) problem. For each state, the three numbers in parentheses are the current content of Jugs A, B, C, respectively. S, T, etc., are the state labels.

Figure 2.1: Problem space for the (8, 5, 3) Water-Jug problem from the 1980 work by Atwood, Masson and Polson [7].

2.1.5 Sources of Subjective Difficulty

Early problem-solving studies using this computational perspective drew upon a number of problems of transformation, including the well-known Water-Jug, Towers of Hanoi, Hobbits and Orcs, and Missionary and Cannibals problems (See, e.g., [7, 8, 33, 39, 46, 51, 71, 105, 103, 107, 117]), as well as some lesser known serial pattern problems and maze tracing problems [62, 104]. From these studies, a number of theories emerged to try to explain the source of the difficulty of problems of transformation.

The difficulty of a problem is dependent on a number of factors, including the interaction between the problem solver and the problem [66] and on the objective difficulty of the task itself. In this section the focus is on the difficulty of problems from the perspective of the interaction between the problem solver and the problem, that is, on a subjective measure of difficulty. The field of computational complexity has at its disposal a wealth of tools for analyzing the objective difficulty of problems of this type, and the objective difficulty of these problems will be addressed later in Section 2.2.

People often encounter problems that seem difficult, but formalizing what actually makes these problems difficult can be a challenge. Newell and Simon suggested that the difficulty of a problem is related to the size of a problem's problem space [66]. Indeed, known hard problems like chess have exponentially large problem spaces [86, 87]. But the size of a problem's problem space is not necessarily sufficient to determine its difficulty. In the terms that have already been defined, the difficulty of a problem stems from overcoming the obstacle that impedes the path to the goal, or choosing the correct sequence of operations that will lead to the goal. A number of factors have been identified that appear to impact performance on problems of transformation, and therefore may impact their difficulty. These factors include the specificity of the goal, the order in which problems are presented, and schema acquisition. These factors are all independent of the problem in question and are believed to be general factors that can impact performance on problems of transformation.

Goal specificity has been found to contribute to the difficulty of some problems of transformation, resulting in better performance when no goal location information is given than when the goal location is known [104]. Information about the goal's relative location to some intermediate states can lead to incorrect operator choices when the path to the goal is circuitous. In effect, problem solvers may be unable to recognize

properties of intermediate states to determine the direction to the goal, incorrectly interpreting confounding direction information. In this case the relative goal location information, while globally correct, is locally incorrect at some intermediate states of the problem, and as a result participants incorrectly infer direction information at these intermediate states and make incorrect move selections.

Another factor that has been found to impact performance, or the subjective difficulty of problems, is the order in which problems or instances of problems are presented. For example, the Einstellung effect has been observed on a number of problems of transformation. Interaction with instances that can be solved with one, often more complex, sequence of moves interferes with a problem solver's ability to recognize when another, perhaps simpler, sequence of moves can be applied to achieve the goal [51, 62, 103]. One possible interpretation of these results is that problem solvers, given problems solvable solely by a particular sequence of operations, tend to infer that sequence is the only correct one, which inhibits their ability to search for a different sequence of moves on subsequent problems for which that sequence may or may not apply.

Concept attainment and learning can take place in parallel with problem solving, and impact performance on performance on subsequent instances. Faced with a novel problem for which no strategy is known and lacking domain-specific knowledge, a novice problem solver will likely resort to a general problem-solving strategy. General problem strategies could include exhaustive search, trial and error, hill climbing, or means-ends analysis [34, 66]. As an individual gains more experience and domain-specific knowledge, more sophisticated strategies may emerge that allow finding a solution to some instances of a problem more efficiently. Differences in problem-solving performance between novices and experts, both in terms of speed and accuracy, have been attributed to experts' ability to quickly and accurately identify correct next moves, even on problems with enormous problem spaces, like chess [49, 102]. One explanation for this difference is that experts possess cognitive structures, called schemata, that allow them to recognize a problem, or instance of a problem, as belonging to a particular class. This classification makes it possible to quickly retrieve appropriate moves or operators. Expert schema acquisition occurs over many interactions with instances of a problem, amounting to the accumulation of rules about what next moves are best given a particular configuration or problem state [28, 97]. Schema acquisition is not limited to experts and has also been observed in novices during a single problem-solving session on a series of related physics

problems [4], and the Water-Jug problem [16]. In novices, the process of schema acquisition can be conceptually broken down into two stages: strategy acquisition, and subsequent schema acquisition. Strategies may also be acquired without associated schema acquisition for a number of possible reasons.

These examples highlight some factors that affect the relative difficulty of a set of problems. At roughly the same time that these efforts were taking place, scholars in the field of computer science were working towards defining a classification of computational problems, based on quantifiable measures of difficulty, that would result in a set of well-defined complexity classes [19, 29]. This classification scheme resulted in an objective measure of problem difficulty, and not surprisingly, some cognitive scientists began drawing from some of these known *hard* problems to extend their understanding of human problem solving. The following section begins with an overview of how problems are classified based on their complexity in the field of computer science, and then moves on to describe human performance studies on a handful of problems that have been determined to be hard, based on these complexity classes.

2.2 Computational Complexity

In many ways, problem solving from the point of view of computer science is very different from the problem solving discussed thus far. When studying human problem solving, participants are given a small number of instances of a problem, often small in size, and are asked to find solutions to them. Solving a problem amounts to getting (possibly) correct solutions to very few instances of a problem. In contrast, when a computer scientist solves a problem, it typically means they have found a general method that always finds a correct answer to any instance of that problem.

This distinction may make it seem as though there is no real need to take this computational perspective into consideration if human problem solving is of interest, rather than general algorithm design. However, given that humans face hard problems frequently, as well as easy ones, it is important to define, as best possible, what it means for a problem to be easy or hard. Furthermore, it is prudent to take both the objective and subjective measures of problem difficulty into consideration when trying to interpret human problem-solving performance. The study of computational complexity provides one such objective measure of difficulty, and therefore complements the subjective measures of difficulty reviewed thus far.

2.2.1 Computational Problems

While problems of transformation have sufficed thus far, computer scientists typically work with another kind of problem: a *computational problem*. A computational problem is an input-output mapping, typically specified as follows:

PROBLEM NAME
Input: Describes a given instance i from a general class of problem I.
Output: Describes the required output, $\Pi(i)$.

A computational problem can also be thought of as two sets: a set of instances of a problem and an associated set of solutions for each instance. The definition of a computational problem describes the problem in general, for every possible instance, whereas for the problems described above in Section 2.1.2, a problem usually referred to a specific instance of a problem.

2.2.2 An Instance versus a Problem

At this junction, it becomes important to address the difference between *an instance* of a problem, and *a problem*. A problem is defined as having an input and an output, in general terms. An instance of a problem, on the other hand, is specified in terms of actual values of the input. To illustrate, consider the 3-Water-Jug problem discussed earlier. This problem of transformation can also be defined as a computational problem, as follows:

3-WATER-JUG PROBLEM (search version)
Input: Three jugs, (A, B, C) a capacity c_A, c_B, c_C for each jug, a starting fill-level of water for each jug $S = \{s_A, s_B, s_C\}$, and a goal fill-level of water for all three jugs $G = \{g_A, g_B, g_C\}$.
Output: A sequence of legal pouring steps that transforms S to G. A legal pouring step involves transferring water from one jug to another, such that the overall volume of water in the three jugs remains constant.

An instance of this problem, in contrast, is given by specifying actual values for the variables given in the definition. For example, the instance of this problem described above could equivalently be defined as: ($c_A = 8$, $c_B = 5$, $c_C = 3$), ($s_A = 8$, $s_B = 0$, $s_C = 0$), and ($g_A = 4$, $g_B = 4$, $g_C = 0$).

This distinction is important when discussing the complexity of a problem versus the complexity of an instance. In particular, an algorithm that *solves a problem* must by definition solve every instance of that problem; however, an algorithm that *solves an instance* of a problem need not solve the problem in general. The difficulty of these two tasks can differ greatly.

To illustrate the implications of this difference, consider the problem of multiplying two positive integers: $x \times y$. A simple algorithm to solve this problem is to convert the problem to addition, by adding x to itself y times: $4 \times 3 = 4 + 4 + 4$. This algorithm will work for any two positive integers. Now consider the following instance of this problem: 9×10. An algorithm that will solve this instance of the problem, and one that is commonly used to solve a subset of instances of this problem that follow this pattern, is: place the digit 0 after the digit x. This algorithm will work for the instance where $x = 9$ and $y = 10$ to yield 90, but will fail on many instances, including the one where $x = 4$ and $y = 3$ (it would yield 40, which is not correct).

The importance of this distinction has repercussions throughout this document, in particular when discussing how a problem solver works with an instance of a problem. Further, this distinction is completely overlooked in most human problem-solving literature, where the term *problem* is used in the place of both *problem* and *instance*. Hereafter in this work, the term problem will be used to describe a problem in general, and the term instance will be used when specifying an instance of a problem with actual values.

2.2.3 Time Complexity Primer

In the field of computer science, or more specifically in the study of computational complexity, computational problems are analyzed based on various measures such as memory requirements or time requirements. We concern ourselves with the latter, measured in terms of the size of the input. Classifying a problem based on this measure generalizes the amount of time required to find a solution *for any given input*, and allows for problems to be compared in terms of difficulty, regardless of specifics of their given input. This method, known as *time complexity analysis* classifies a problem based on the rate at which the number of steps needed to find a solution grows (worst case), relative to the size of the input. Some rates of growth are considered *slow* and therefore reasonable, like linear $f(n) = 2n$ or polynomial $f(n) = n^2$, while others are considered *fast* and therefore infeasible, like exponential $f(n) = 2^n$. For small

values of n, linear and polynomial growth differ (see Figure 2.2), but at moderate sizes are essentially indistinguishable (see Figure 2.3), whereas exponential becomes large very quickly. For this reason, growth rates are typically classified with polynomial or slower time together, and those that are exponential (or worse!) together.

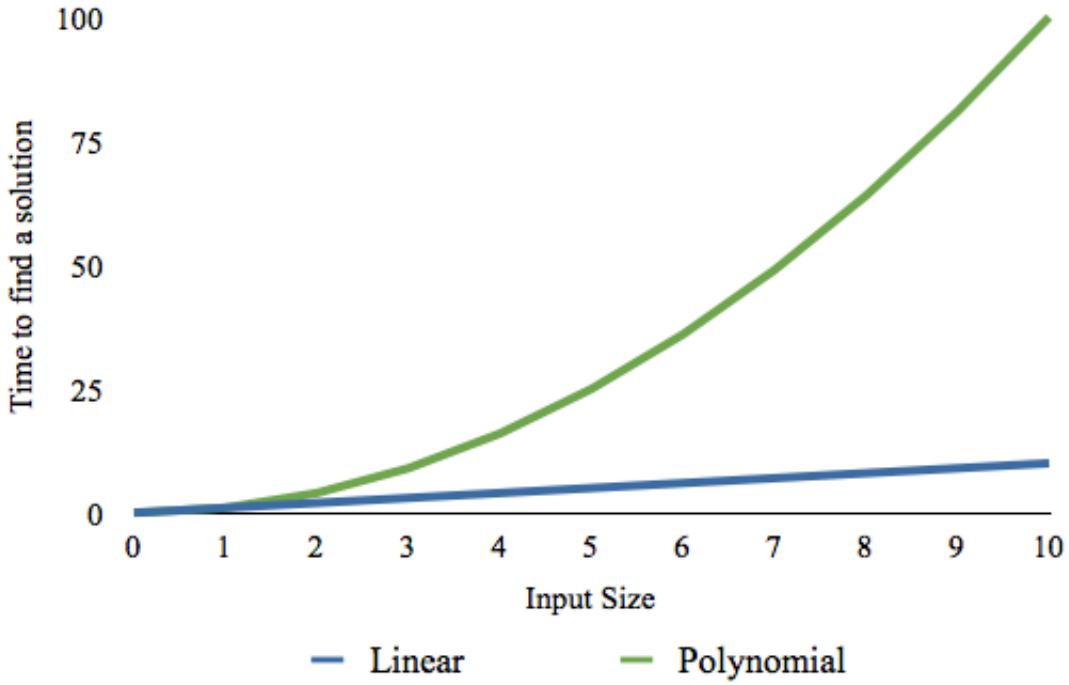


Figure 2.2: Visual comparison of linear and polynomial growth rates of functions. Linear time is shown in blue, and polynomial time is shown in green.

To date, a great number of problems have been classified into various complexity classes. In this section some of the time-complexity classes used to classify problems based on how much time (number of steps) is required (worst case) to solve the problem as a function of the input size are reviewed.

2.2.4 Graph Problems

Throughout this work a number of graph problems will be presented. A *graph* is a commonly used mathematical construct, used to represent a set of objects and a set of relationships between these objects. The objects are often referred to as *vertices* (singular: *vertex*), and the relationships are referred to as *edges*. Edges can be

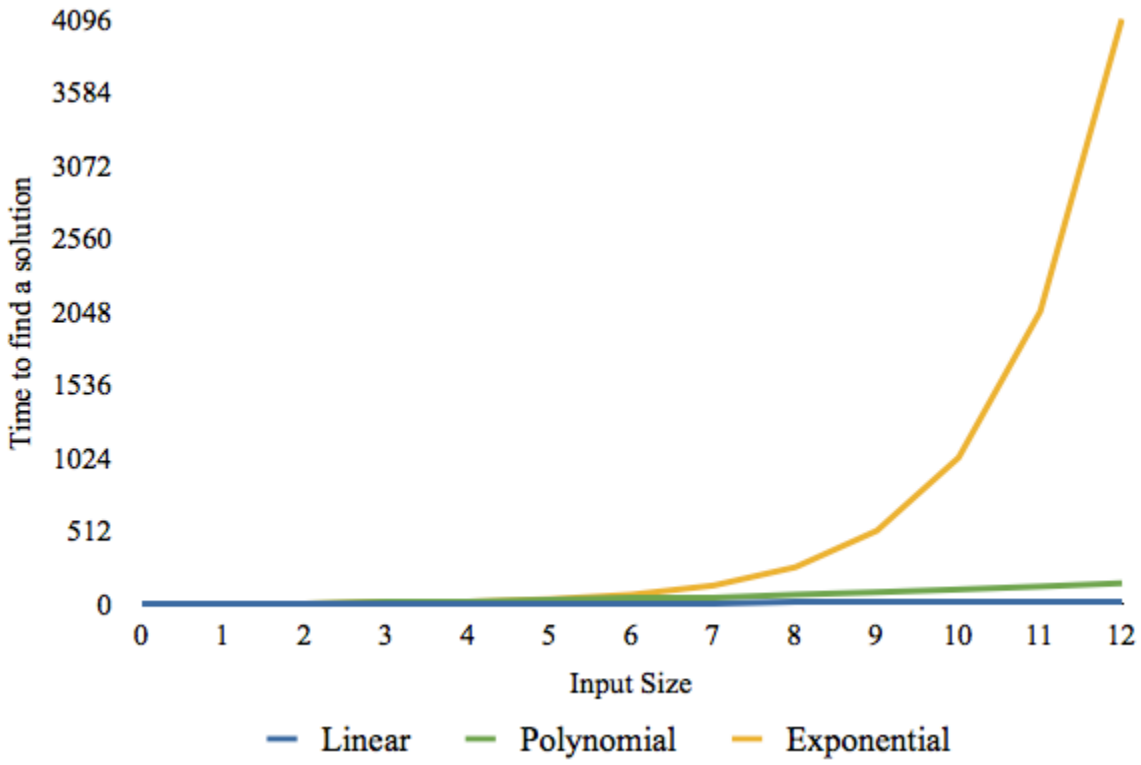


Figure 2.3: Visual comparison of linear, polynomial, and exponential growth rates of functions. Linear time is shown in blue, polynomial time is shown in green, and exponential time is shown in yellow.

directional when they refer to one-way relationships, or *bidirectional* when they refer to reciprocal relationships. Graphs are said to be *directed* if they contain directional edges (see Figure 2.4), or *undirected* if all edges are bidirectional (see Figure 2.5). A *family tree* is an example of a directed graph, where the vertices represent people, and edges represent relationships with offspring. A communications network can be thought of as an undirected network, where information can travel between any two locations in the network connected by a communication line or wireless signal. For the remainder of this work, unless noted otherwise, all graphs are undirected.

Graphs provide a natural way to visually represent complex relationships and connections between objects, and are even readily understood by young students with little or no experience with formal graph theory education [14]. Graphs are a powerful abstraction tool, allowing complex relationships to be presented in a simple

way. As such they are a natural choice for presenting problem-solving tasks in a visual manner.

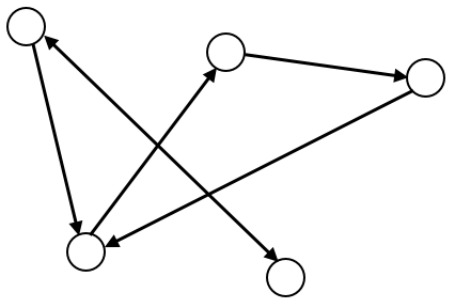


Figure 2.4: A simple directed graph. Vertices (dots) represent objects, and directed edges (lines) represent directional relationships between connected vertices.

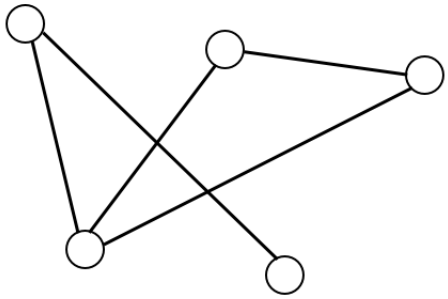


Figure 2.5: A simple undirected graph. Vertices (dots) represent objects, and edges (lines) represent reciprocal relationships between connected vertices.

2.2.5 What is an Algorithm?

Another important concept in computer science is that of an *algorithm*. An algorithm consists of a set of unambiguous operations or steps [20], and is said to *solve a computational problem* if, and only if, on every instance of the problem it always yields a correct solution and always terminates. This differs from what is typically meant in the study of human problem solving where the phrase *solve a problem* is used to mean *find a way to overcome an obstacle to the goal*, rather than *find a way*

to always find a correct answer. This distinction will be the topic of more in-depth discussion in Section 2.3.2 below, but for now it is sufficient to keep in mind that there are different possible interpretations of this phrase or idea.

To start, some of best known complexity classes are presented: *P*, *NP*, and *NP-complete*. Informally these classes represent problems that are, respectively easy to solve, solvable, and hard to solve.

2.2.6 Class P

The class P consists of all computational problems for which an algorithm exists that runs in at most polynomial time (in terms of the input size) for any given input. These are problems that can be solved, exactly, in a reasonable amount of time, and are said to be *tractable*.

The problem of finding the largest number in a list of n integers, is in the class P. An algorithm that solves this problem could be described as follows. Pick the first item in the list, set it to the current maximum, then iterate through all items in the list. At each step, compare the current item to the current maximum, replacing the current maximum if the current item is greater. When all items have been visited, return the current maximum. This algorithm runs in linear time in terms of the input size. There are n items in the list and each one is compared to the current maximum, taking n steps. Therefore this problem is in the class P.

A well-known, and arguably more interesting computational problem in the class P is the Minimum Spanning Tree problem (MST). This problem asks, given a graph, for a spanning tree, that is, a set of edges that connects all vertices in a given graph with no cycles, and of least cost. This is akin to the task of connecting a set of cities by roads, so that every city can be reached from every other, with the minimum length of road required.

Minimum Spanning Tree Problem (decision version)

Input: A graph $G = (E, V)$, with vertex set V, and edge set E, and a set of costs, or edge-weights, C, where c_i is the cost of edge e_i.

Output: Yes/no answer to the question: does there exist a subgraph G' of G that is a spanning tree, such that all $v \in V$ are connected in G', where $\sum_{i=1}^{n} c_i$, $e_i \in G'$ is minimized?

This problem is in the class P because there is a polynomial-time algorithm that can find a minimum spanning tree of any graph G. Kruskal's Algorithm [44] is one

such algorithm. Informally, this algorithm consists of building up the spanning tree by adding the cheapest edge, without creating a cycle. It can be described as follows:

- initially all vertices in G are in G', and no edges from E are in G';
- until all vertices in V are connected in G'
  - select the edge e_i of least cost that does not create a cycle in G'
  - add it to G'.

To see why this only requires polynomial time, let us break down the required steps in terms of the input size (the input size of a graph is typically quantified by the number of vertices in the graph). For $|V| = n$, then $|E| = e$ is bounded by n^2[1]. The total time needed to build G' is $O(n^2)$. One can build G' by first sorting all edges in the graph by cost in $O(n^2 log(n))$ time [20] and then add $n-1$ edges to the tree in $O(n^2)$ time. Recall from Section 2.2.3 that this running time is considered tractable. See Appendix A.1 for details about *big O notation* and lower bounds on running times.

While polynomial time is typically used as an indication of a *reasonable* running time, some polynomial running times (i.e., n^{100}) may not be feasibly implemented by a computer or by a human. However, anecdotally at least, many computational problems in the class P are known to have fast polynomial running times.

A class of problems has now been identified, those in P, that can be considered feasible. The upcoming sections consider how to show that a problem is infeasible (at least in terms of time-complexity), which first requires us to define a very important class of problems: the class *NP*.

2.2.7 Class NP

The class *NP* contains problems for which an algorithm exists, and whose solutions can be checked in polynomial time. This means, given an instance and a candidate solution, it is possible to check the veracity of the solution in polynomial time. Informally, these are problems that can be solved, but unlike problems in P, without the requirement of knowing how to solve them quickly. These are also problems for which, if given a solution, it is possible to check the validity of that solution in a

[1] In a graph with n vertices, each vertex can be connected to at most every other vertex. The number of edges therefore cannot exceed $(n-1) + (n-2) + ...(1)$ if edges are counted once each.

reasonable time. This does not, however, mean that finding that solution is easy. There is a common misconception that the Class P means *polynomial*, and therefore that the Class NP means *non-polynomial*. This is not, as shall be shown, the case.

An example of a problem in the class NP is the Vertex Cover problem. The Vertex Cover problem asks, given a graph, for a subset of vertices that covers all the edges in the graph. An edge is covered by the vertices to which it is connected[2]. More formally, this problem can be defined as:

Vertex Cover / Vertex Cover (decision version)

Input: A graph $G = (E, V)$, with vertex set V, edge set E, and a positive integer k.

Output: The *YES/NO*-answer to the question: Does there exist a vertex cover of size at most k? That is, does there exist a subset U of V of size k, such that every edge in E is incident to at least one vertex in U?

As an example, a vertex cover of size seven is shown in Figure 2.7 for the graph in Figure 2.6. The seven vertices coloured blue constitute a valid vertex cover, since every edge in the graph is incident to at least one of these vertices.

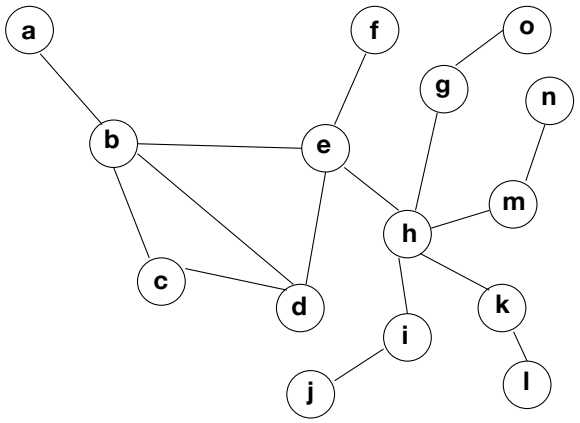


Figure 2.6: A Graph G with 15 vertices.

To prove that a problem Π is member of *NP* it must be shown that:

- an algorithm exists for Π, and

[2]A more informal way to present this problem is to state that if the graph represents art galleries, where the vertices were guard posts and the edges were the hallways containing priceless art, the problem is to determine if it is possible to cover all the hallways with at most a given number of guards.

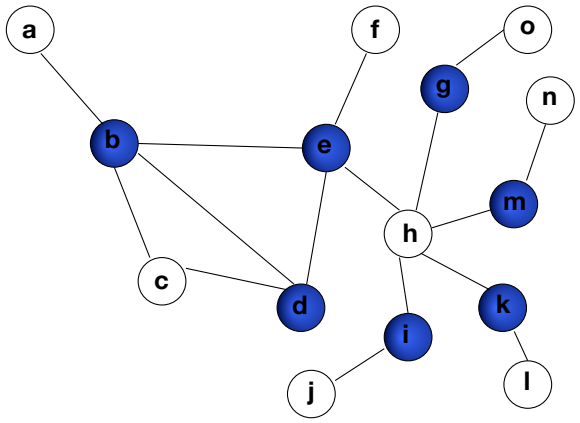


Figure 2.7: A valid vertex cover of size 7, for G, is shown in blue. This vertex cover is valid because every edge in G is incident to at least one vertex in the cover.

- its solutions can be verified in polynomial time.

Let us show that the Vertex Cover problem is in the Class NP.

A brute-force algorithm can be used to find a valid cover (if one exists) of size k for G. Since the running time of this algorithm is not of concern, but rather only that one exists, it is possible to try every single possible set of vertices of size k, and check if it is a valid cover. If none exists, then the answer to the decision problem is NO, otherwise the answer is YES. In order to be shown to be a member of NP it must be shown that a polynomial-time algorithm exists that verifies a solution. Consider, for example, an instance of the k-Vertex Cover problem on a graph $G = (V, E)$, and a candidate solution, $V' \subset V$. To check whether V' is a valid solution, one must check that:

1. $|V'| \leq k$, and
2. that every edge in E is incident to at least one vertex in V'.

The first step can easily be done in time linear to the size of V. The second step requires iterating through each edge in E and checking if at least one of its end-points is in V'. The number of edges in E is bounded by $O(n^2)$ (as each vertex can have at most $(n-1)$ neighbours, resulting in at most $(n-1) \times (n-1)$ edges). Therefore, worst-case, the second step would require $O(n^3)$ time. The overall running time of this algorithm is $O(n^3)$, which is polynomial, and therefore Vertex Cover is in the class NP.

Another problem in the class NP is the Travelling Salesperson problem (TSP), which is of interest here as it has been studied extensively in the context of human problem solving. In the Travelling Salesperson problem, the goal is to find the cheapest or shortest tour that visits every city (or point) exactly once, starting and finishing at the same location. One can imagine a salesperson who starts at their home town, visits all towns in his/her region, then returns home. The goal is to minimize the cost, be it distance, fuel cost, or time. This problem has applications in routing and transportation, but also in circuit design and construction [48].

Traveling Salesperson Problem (TSP) (optimization version)

Input: A finite set of cities, $C = \{c_1, c_2, ..., c_n\}$, a distance, $d(c_i, c_j) \in Z^+$ for each city-pair $c_i, c_j \in C$.

Output: A tour of all cities in C with minimum length. In other words, an ordering $< c_{\pi(i)}, c_{\pi(2)}, ..., c_{\pi(n)} >$ of C, such that $\sum_{i=1}^{n-1}(d(c_{\pi(i)}, c_{\pi(i+1)})) + d(c_{\pi(n)}, c_{\pi(1))}$ is minimized.

A variant of the TSP, the Euclidean Travelling Salesperson problem, or E-TSP, restricts the vertices to being in the Euclidean plane, and the associated costs of the edges to the Euclidean distance between the points. If these distances are restricted to a fixed level of precision, then this problem is also in the class NP.

Euclidean Traveling Salesperson Problem (E-TSP) (optimization version)

Input: A finite set of points in the plane, $P = \{p_1, p_2, ..., p_n\}$, where the a distance, $d(p_i, p_j) \in R^+$ for each city-pair $p_i, p_j \in P$ is the Euclidean distance between p_i, p_j.

Output: A tour of all points in P with minimum length. In other words, an ordering $< p_{\pi(i)}, p_{\pi(2)}, ..., p_{\pi(n)} >$ of C, such that $\sum_{i=1}^{n-1}(d(p_{\pi(i)}, p_{\pi(i+1)})) + d(p_{\pi(n)}, p_{\pi(1))}$ is minimized.

2.2.8 The Relationship between P and NP

Two very important class of problems have been defined, that can be thought of informally as those with algorithms (Class NP) and those with fast algorithms (Class P). But what is of interest here is how to classify problems as difficult, that is, those problems that have algorithms but for which no fast (polynomial-time) algorithm exists. It turns out that this is not a simple task. The open question of how the

classes P and NP are related, is one of the Millennium Prize problems, and solving it is worth $1 million[3].

What is known is that P must be contained in NP, as any problem that can be solved in polynomial time, can also have its answer checked in polynomial time. Therefore P is a subset of NP. However, what is not known is whether P is a proper subset of NP ($P \neq NP$). If P is a proper subset of NP then there exists some problem in NP that is not in P, implying that there exist problems whose solutions are checkable in polynomial time for which no polynomial algorithm exists. For a visual representation of the two possibilities (either $P \neq NP$, or $P = NP$), see Figure 2.8. If, on the other hand, $P = NP$, then all problems for which algorithms exist must have fast (not super-polynomial) running times.

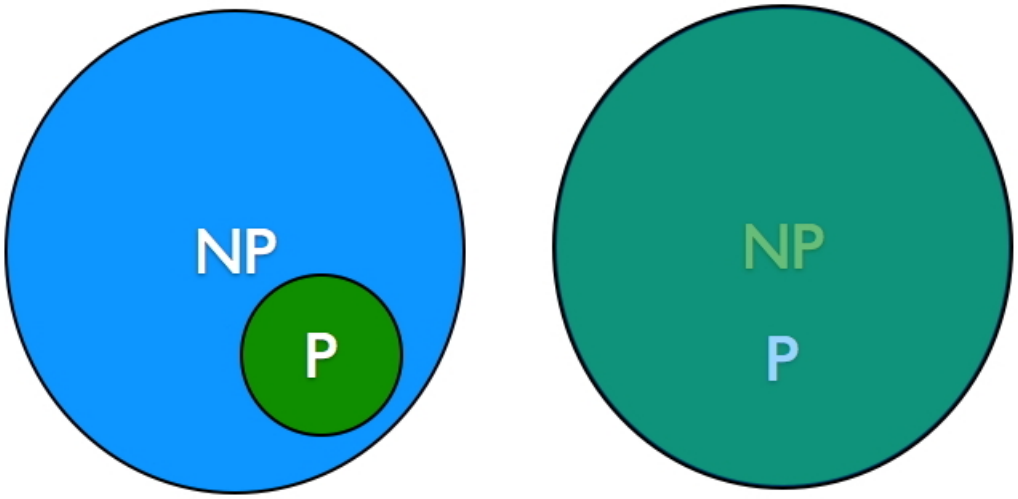


Figure 2.8: P not equal to NP shown on the left, and P equals NP shown on the right.

It is commonly assumed that $P \neq NP$, that is, that there are some decidable problems that are hard, in the sense that they require an amount of time to solve that is super-polynomial, which is considered (in the terms of classical complexity) an unreasonable amount of time. This shall be assumed henceforth, and based on this assumption, the class *NP-complete* is a class of problems that can be considered hard.

[3]The Clay Institute offers 1 million US dollars for solutions to any of its Millennium Prize Problems, including the question of $\mathcal{P} \neq \mathcal{NP}$ http://www.claymath.org/millennium/.

2.2.9 Class *NP-complete*

Informally, the class *NP-complete* contains all the hardest problems in NP: those problems that have an algorithm, but for which it is believed that no polynomial time algorithm exists. These are problems that are considered *intractable*.

To show that a problem Π is *NP-complete*, one must show:

1. that Π is in *NP*, and
2. a polynomial time reduction from a known *NP-complete* problem to Π [5].

A polynomial-time reduction is used to show that Π could be used to solve all the other *NP-complete* problems. This implies that if Π is easy to solve, then so are all other *NP-complete* problems. This is a consequence of the fact that (except for *SAT*, the first problem shown to be *NP-complete* [5]) the hardness of all *NP-complete* problems is based on each problem being proven to be at least as hard as any other *NP-complete* problem: their hardnesses are interdependent. The reduction involves describing how a known *NP-complete* problem could easily be transformed into Π, or reduced to Π.

Formally, a problem Π' is *reduced* to another problem Π if there exists an algorithm that transforms Π' into Π. This reduction can be used to show that Π is at least as hard as Π'. You start with a problem known to be hard and transform it into the problem in question.

A polynomial-time reduction is one for which the time needed for the transformation is bounded by some polynomial, n^c for c constant, and n the size of the input. The implication is that there is mechanism that only requires polynomial time to use Π as a subroutine that will solve a problem known to be difficult, Π'. This means that the transformation is *easy*. If a polynomial-time algorithm is found for a single *NP-complete* problem, then all *NP-complete* problems would have polynomial-time algorithms, essentially collapsing NP down to P. Despite a great deal of effort by mathematicians and computer scientists, no such algorithm has been found. In addition, it has yet to be proven that there does, or does not, exist a poly-time algorithm for an *NP-complete* problem.

Of the problems introduced so far, Vertex Cover, TSP, and E-TSP have been determined to be in the class *NP-complete*[4] [29, 74]. In addition, the games and

[4]A variant of TSP and E-TSP is used to show membership in the class *NP-complete*, that will be discussed later in Section 2.3.3

puzzles described above in Section 2 have also been shown to be *NP-complete* (or in other complexity classes beyond the scope of this work), including: Tetris, Mastermind, Rush Hour, Minesweeper, Super Mario Bros., Donkey Kong, Legend of Zelda and Pokemon [2, 3, 22, 42, 81, 88, 89, 100]. Of the everyday hard tasks described above, a number are also very closely related to problems that have been proven hard. Scheduling tasks to minimize conflicts, for example, can be equivalent to the Vertex Cover problem; packing the most items in a container is akin to the Bin Packing problem that is known to be hard; and finding the best way to satisfy a number of constraints is much like SAT that is known to be hard [5, 29]. In addition, a variant of the Water-Jug problem, with n jugs where jug i has capacity c_i, in which the problem solver may fill any jug from a tap (to capacity only), dump any jug out down the drain (in its entirety), or pour from one jug to another (with the total volume remaining constant) is hard [7, 90]. It is an open question if this hardness result extends to the limited k-Water-Jug problem (the generalized version of the 3-Water-Jug problem with k jugs).

2.2.10 Computational Approaches to Coping with Complexity

Many hard computational problems still have real world applications and therefore once a problem is found to be hard, either in the class *NP-complete* or another hard complexity class, work typically turns to finding alternative ways of finding possibly non-optimal solutions. This includes finding approximation algorithms, heuristics, or identifying properties of the problem that can allow finding exact solutions quickly under particular constraints. One of the reasons for addressing this approach here is that the terms approximation and heuristic have very specific meanings in this context, which differ from how they are used in the field of cognitive science. So, in order to avoid misinterpretation, the computer science definitions are introduced here.

Approximation Algorithms

Colloquially speaking, the term approximation is usually taken to mean finding an answer that is close to correct. Computer scientists have a more precise definition when referring to approximation algorithms. Typically, an approximation algorithm must be guaranteed to find a solution within a specified bound of correct. Some

NP-complete problems, despite their intrinsic difficulty, can be found to have approximation algorithms that are bounded by a constant factor.

More formally, a problem Π is said to have a ρ-approximation algorithm A with cost $f(x)$ if, on any input x, the approximate solution $A(x)$ is bounded by a factor ρ times the value of an optimal solution. A problem is said to have a polynomial-time approximation scheme (PTAS) if there exists a polynomial time algorithm that will always produce a solution that is within a factor $1 + \epsilon$ (for a minimization problem) of the cost of the optimal solution, or $1 - \epsilon$ (for a maximization problem) where $\epsilon > 0$. The implication of both of these kinds of approximation is that it is possible to find relatively close to optimal solutions in a reasonable amount of time for problems with these kinds of approximation algorithm, with an added guarantee of a limit of error. It is also possible to prove for some problems that no constant factor approximation algorithm exists [40], and some other approach must be taken if they have a practical use.

Heuristics

For problems that have been shown to not have an approximation algorithm, or for which the bounds of known approximation algorithms may not be good enough for practical applications, one approach is to turn to heuristics. In the study of computational complexity, a heuristic typically refers to a *heuristic algorithm*: an algorithm that, unlike an approximation algorithm, has no guarantee of correctness or quality of bounds for a given computational problem [20]. A heuristic may not always find an exact solution, but is still an algorithm, and therefore consists of an unambiguous set of operations and must halt. In the field of cognitive science, on the other hand, the term *heuristic* has more ambiguous definitions including: "...a strategy that ignores part of the information, with the goal of making decisions more quickly, frugally, and/or accurately than more complex methods" [30]; or a process "serving to discover or find out" [65]. These definitions lack the precision of the computer science definition, with no requirement to either halt or be unambiguous. In order to be clear, when referring to the computational complexity definition of a heuristic, the phrase *heuristic algorithm* will be used, and when referring to the *rule of thumb* definition, the term *heuristic* will be used.

Class FPT

Approximation algorithms and heuristic algorithms provide good ways of dealing with hard problems that still have practical uses however they have the disadvantage of yielding solutions that are not necessarily correct. Fortunately another approach to dealing with hard problems exists, one which can still solve the problem exactly. Furthermore, despite the difficulty of the problem, this approach, if applicable, requires time that is polynomial in terms of the input size n (tractable), and only non-polynomial in terms of some fixed parameter k. The practical implication of this approach is that it somehow captures the difficulty of the problem, the part that blows up the running time, in a small constant rather than the size of the input. This is typically achieved through the process of *kernelization*, in which preprocessing is used to reduce the input to a smaller input, or *kernel*. Solving the kernel, in conjunction with the preprocessing steps, solves the original input. The preprocessing stage often involves the application of one or more *reduction rules* that in essence remove parts of the instance that are easy to deal with. If the application of these reduction rules can be achieved efficiently and the size of the kernel can be bounded, then the problem can be shown to be fixed parameter tractable (FPT) and belong to the class FPT [23].

More formally, a problem Π with input i and parameter k, is said to be Fixed Parameter Tractable, if Π is decidable in $O(f(k)n^c)$ time, where $f(k)$ is a function that depends only on k, for some constant c, and n the size of i. In a sense, the *bad* aspect of the running time is captured solely in $f(k)$, and does not grow with n.

The implication of a problem being in the class FPT is that one can identify an aspect of the problem that is independent of the input size, that captures the *hardness*, and therefore for many instances, a reasonable running time algorithm exists. Some decision and search problems, for instance, take as input a parameter that is not dependent on the input size. Consider, for example, the decision version of Vertex Cover that includes the parameter k, the size of the desired vertex cover. This is a natural parameter for a FPT result. For small enough k, the resulting running time is no different from polynomial, as the exponential portion is captured entirely in terms of this parameter.

The stage is now set to proceed to look at the current state of research on human performance on *difficult* problems: studies of human performance on *NP-complete* problems.

2.3 Human Performance on Hard Computational Problems

While the focus of these work is on human performance on hard computational problems, current research has focused on hard optimization problems, which is reviewed here. A number of factors have been claimed to impact performance on these, and other problems of transformation, which will also be reviewed and discussed. Finally, a different kind of problem is presented, one which is as well-suited, if not better, than hard optimization version of problems which have been the focus to date for studying human performance on hard problems.

2.3.1 Human Performance on Hard Optimization Problems

In the 1980's, some efforts to further the study of human problem solving turned attention to hard computational problems. These known hard problem in many ways are ideal candidates to help better understand how the human cognitive system might cope with complexity. They are problems that are provably difficult, and yet easy to represent. This work focused primarily on variations of the popular Traveling Salesperson problem (TSP), defined earlier in Section 2.3.3. Variations of TSP that have been studied in the context of human performance include: E-TSP, E-TSP with obstacles [36], TSP using a city-block configuration [116], and 3D E-TSP [35]. The problem has been presented visually on paper (See, e.g., [52, 54, 55, 114, 106, 116]) and on a computer screen (See, e.g., [10, 24, 25, 32, 45, 72, 85]), virtually using a simulator, and in a real-world interactive environment using objects placed on a floor [35]. In general, studies claim that humans typically find solutions that are close to optimal (See, e.g., [32, 72, 112, 116]), taking approximately linear (in terms of the number of points in the input) time to complete [24, 25, 32].

In addition to TSP and its variants, a few other hard computational problems have been used in a limited number of human performance studies. These include: the n-Puzzle problem [76], the Generalized Steiner Tree problem [11], the Knapsack problem [37], and the Minimum Vertex Cover problem [13, 37]. In the n-Puzzle study conducted by Pizlo and Li [76], participants were tasked with finding the shortest sequence of tile movements needed to arrange the tiles in numerical order. Participants' solutions to instances of this problem ranged in quality, and were not believed to be close to optimal, ranging from 40 to 160 moves however, optimal solution-length

was not known for all instances presented. Relative solution quality was based on results of optimal solution length on randomly generated instances, which has found to be between 41 and 66 states. One study of the Euclidean Steiner Tree problem looked at possible correlations between performance on this, and other computational problems, and general intelligence measures, however no specific performance results were discussed [11]. In their study of the problem-solving strategies used on the optimization version of Knapsack and Vertex Cover problems, Hidalgo-Herrero et al. found that most adults and children were able to find optimal solutions to many instances of both problems [37]. A study of the Minimum Vertex Cover problem [13] focused on properties of instances that impact human performance, and what strategies participants might adopt when tackling instances. Performance on instances of this problem was in line with performance results reported on the E-TSP, ranging, depending on the factor, from roughly 4% to 10% above optimal.

Many of these findings are seemingly at odds with the current understanding of the complexity of these problems. These problems are hard, and therefore no fast algorithm likely exists that can solve them. There are also strict bounds on how close approximation algorithms can get to the optimal path of TSP [41], and these approximation algorithms typically require polynomial time to run, in contrast with the *linear time* mentioned above.

Despite these seemingly excellent performance results, a number of factors appear to impact the difficulty of some instances of these hard computational problems.

2.3.2 Factors that Impact Performance

Impact of Instance Size

Conventional time-complexity results of course take the size of the instance directly into account, as these time-complexity results relate the amount of time needed to find a solutions to the size of the input. If the instances presented to participants are too small, it may be possible to use exhaustive search, or some other strategy that would not be feasible on large instances, to find a solution in a reasonable amount of time. However, for hard computational problems, this option becomes infeasible for even seemingly small instances. For E-TSP, for example, the number of possible tours on a set of n points is bounded by $f(n) = n!$, or $O(n!)$, which for $n = 10$ is over 3-million tours. Participants in these studies are typically given instances ranging in size from 10 to 40 nodes, that are likely too big for any type of brute force approach

to be feasible. Additionally, the near-linear solution times observed in a number of studies contradicts the possibility that exhaustive search can explain performance results. Performance on hard computational problems tends to get worse as instances get larger, which is to be expected.

Impact of Instance Selection

In some studies, instances were selected or generated in order to manipulate their relative difficulty. Instances have been generated purposely to include a specific feature or property [54, 55], generated at random given some constraint (number of vertices, or density) [32, 55], generated to manipulate the symmetry of the input or the strategies that are applicable [13, 37], or even generated based on real-world data [55]. Variation in performance results between different types of instances supports the hypothesis that some instances of even these very hard problems are easier than others and highlights the need to be prudent about how instances are generated. This in turn has implications in interpreting performance results. General complexity results are not necessarily representative of the difficulty of specific instances, and therefore instance selection has serious implications on how results ought to be interpreted.

For E-TSP a handful of instances have been identified that appear to interact with the human visual system and render them more difficult than other instances of the same size [54]. It is theorized that the Gestalt principles of continuation and good form may drive tour selection, and as a result it is possible to purposely generate instances which trick problem solvers into finding relatively poor tours.

Impact of Perceptual Processes

Great variation in performance results has been found between even relatively small instances of E-TSP by purposely manufacturing instances [106, 112]. It appears that Gestalt principles of good form and continuation [106, 118] can drive the selection of paths during the process of searching for an optimal tour, quite frequently with a positive impact on performance. However, careful selection of vertex placement can also lead to highly suboptimal tours, as the principle of continuation appears to interfere with the goal of optimization. These results suggest that subjects are able to leverage, perhaps without being aware of it, powerful perceptual processes, again highlighting the importance of being cautious in selecting instances that are representative of the general complexity of the problem. This is an example of a

factor that has been found to impact performance on instances to a problem, that may not be problem specific.

Problem Solving versus Solving a Problem

A number of factors may impact the relative difficulty of specific instances of these hard computational problems. Before proceeding, it is important to consider another aspect that can impact performance, one that has been alluded to in previous sections.

In the study of human problem solving when a person has *solved a problem*, what is often meant is that the person has completed an instance in such a way that they are satisfied with the solution. There is no requirement that the solution in fact be correct, because, unlike an algorithm, people are not expected to always find perfectly correct solutions. It is perhaps tempting to say that this vague definition of problem solving is sufficient when dealing with human problem solving. However, this definition is no longer acceptable, for instance, once computational complexity is used as a measure of hardness, because of the underlying assumption that the problem solver is working on an instance of the problem for which the complexity being claimed is known, rather than some other, perhaps easier, problem. If the phrase *solve a problem* is taken to strictly mean that a participant found a correct solution to an instance of the problem, then this would be acceptable. However, if the phrase is taken to mean the participant arrived at a state other than a valid goal state and chose to find it satisfactory, then the phrase is being used ambiguously. In order to be able to make meaningful interpretations of problem-solving performance, it is vital to understand what problem is being solved, which is in keeping with Marr's perspective that in order to best understand what algorithms the cognitive system might use to solve a computational task, one must first understand what problem is being solved at the computational level.

It is assumed that according to *Marr's Level Theory*, the *cognitive system* can be conceptually divided into three levels (a common assumption, see, e.g., [38, 63, 79, 84, 110, 111]): the *computational level* that contains the cognitive tasks as described by the cognitive functions, the *algorithmic level* that defines the *algorithm* (or set of algorithmic steps) that *solves* the functions, and the *implementation level* that is concerned with cognitive processes that realize these algorithms (see Figure 2.9) [61]. These three levels are often considered independently, and Marr stresses the importance of the computational level, because understanding the nature of the com-

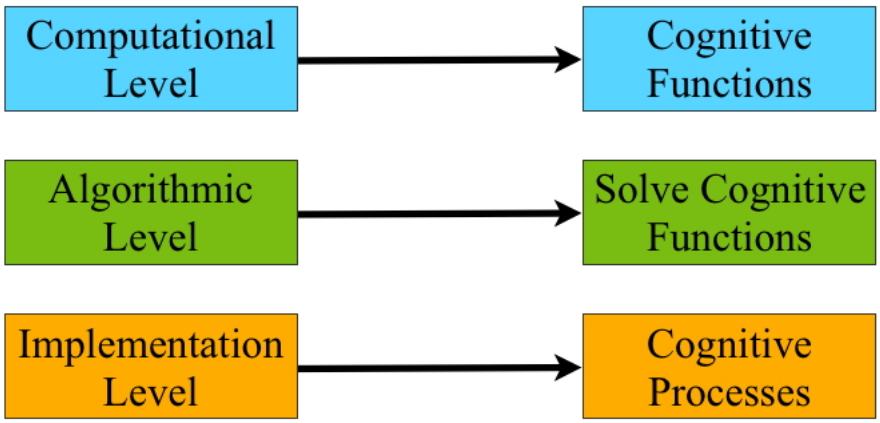


Figure 2.9: Three levels of Marr's Level Theory

putational task will more likely lead to an understanding of an algorithm that will solve it.

When considering human performance results on a given problem, it is assumed that the problem solver is indeed working to solve the given problem, or at least working to solve one or more instances of the problem. On the surface, this assumption seems well-founded. Participants are typically given clear unambiguous instructions prior to being tasked with the problem in question; trials are used to further ensure that participants understand the given task; and computational problems are often quite simple to present, as seen with the problems introduced thus far. Despite these efforts to ensure that participants *comprehend* the problem, it is still very important to ensure that it is possible for participants to solve the given problem. For example, this is the reason for the previously introduced constraint that there exist a path from the start state to the goal.

Terminology used in studies of human performance on hard problems like E-TSP supports the claim that there is an assumption that participants are indeed working on the given task. It is not uncommon for authors to claim that participants are "solving TSPs" [54] when results show that participants' tours are not optimal. Some authors are careful to clarify that tasked with problems that they are not able to solve, participants may still attempt to find a solution, however one that does not match the given goal [32, 112]. For example, van Rooij, Stege and Schactman are careful to state that "[t]his hypothesis states that humans aim to avoid crossed lines in the plane when trying to solve E-TSP because they are sensitive to the fact that tours

with crossed lines are nonoptimal" [112]. Similarly, Graham and Pizlo state "human subjects are quite good at producing near-optimal solutions to the E-TSP" [32]. If it is not possible to be sure that the problem solver is working on the given problem, for example that of finding an optimal tour, then it is vitally important to attempt to determine what problem may be being solved.

There are two (or more) possible solutions to this conundrum. First, if the loose definition of solving a problem is to be taken, then the computational complexity of the task should be discarded.

A second option is to more carefully consider the problem that is being solved by the individual before making performance comparisons based on computational complexity results. In order to include computational complexity as a dimension of difficulty, care must be taken to determine whether or not a given task is encodable by the human system. This does not mean that performance on problems that are perhaps not encodable should not be evaluated, but it does require more explicitly evaluating how the cognitive system, faced with a problem that it cannot meaningfully encode, copes with this issue. This is one of the approaches taken in this work.

2.3.3 Alternatives to Optimization Problems

Altogether, previously discussed results imply that humans are good at finding near-optimal solutions, often surprisingly quickly, to instances of problems that, in general at least, have been proven to be difficult. Instance size and selection have also been found to impact performance, sometimes quite extremely. In the next section, an aspect of how these problems are formulated will be described, one which will be shown to have an even more pronounced impact on performance than anything that has been addressed thus far.

One observation is that human performance studies on hard computational problems have focused exclusively on optimization problems. The optimization version of a problem asks for a *best* solution, either minimizing or maximizing some measure. For instance, the E-TSP asks for a *shortest tour*, the N-Puzzle problem asks for the *fewest moves* that orders the tiles, the Vertex Cover problem asks for the *fewest* number of vertices needed to cover all edges, the Knapsack problem asks for a the *maximum* number of items of at most a given weight, and the Steiner Tree problem asks for *shortest* total length of line segments to connect all points. Optimization has long been a topic of interest for cognitive scientists, and it has been suggested that

some types of optimization problem may exceed the limits of the human cognitive system [94]. An optimization strategy can require the problem solver to store and compare a great number of options in order to determine a solution that is definitively optimal. Given problems that demand too much of the cognitive system, Simon's theory of bounded rationality suggests that people turn to either satisfying heuristics or fast and frugal heuristics [109]. The former, like optimization, works by comparing alternatives until one that is satisfactory is found, the latter using little information or computation. The former most likely applies to hard computational problems like those of interest here.

However, before jumping to the conclusion that faced with hard optimization, problem solvers resort to using heuristics, observe that if optimization exceeds the limits of the cognitive system, then it may be appropriate to study human performance on hard problems that do not include optimization as part of the goal. Fortunately, there are other formulations of computational problems that offer just that possibility. The complexity classes introduced earlier, for instance, are based on the *decision version* of problems, not the optimization versions. The following sections introduce two problem formulations commonly used in the field of computer science, the decision and search versions.

Decision Version

The decision version of a computational problem asks for a yes/no answer to the existence of a solution to the problem. Consider the E-TSP defined earlier, that asked for a shortest tour. The decision version of the E-TSP asks instead if there exists a tour of given length (or shorter).

Euclidean Traveling Salesperson Problem (E-TSP) (decision version)

Input: A finite set of points in the plane, $P = \{p_1, p_2, ..., p_n\}$, where the a distance, $d(p_i, p_j) \in R^+$ for each city-pair $p_i, p_j \in P$ is the Euclidean distance between p_i, p_j, and a bound, $B \in R$.

Output: The YES/NO answer to: Is there a tour of all points in P with length at most B? In other words, is there an ordering $< p_{\pi(i)}, p_{\pi(2)}, ..., p_{\pi(n)} >$ of C, such that $\sum_{i=1}^{n-1}(d(p_{\pi(i)}, p_{\pi(i+1)})) + d(p_{\pi(n)}, p_{\pi(1)}) \leq B$?

The decision version of the Vertex Cover problem, asks if there exists a valid vertex cover with a given number of vertices.

Vertex Cover (decision version)

Input: A graph $G = (E, V)$, with vertex set V, edge set E, and positive integer value k.

Output: The YES/NO answer to: Does there exist a vertex cover for G of size at most k?

The Independent Set problem is closely related to the Vertex Cover problem. This problem asks about the existence of an independent set, that is a set of vertices that are not connected by any edges, of at least a given size[5].

Independent Set (decision version)

Input: A graph $G = (E, V)$, with vertex set V, and edge set E. An integer value k.

Output: The YES/NO answer to: Does there exist a *independent set* for G, of size at least k?

Search Version

The search version of a problem is very similar to the decision version, in that it asks about the existence of a solution that meets a given criterion; however it returns a valid solution if one exists (unlike the decision version to which the answer YES is given) or NO if no solution exists. The search version of Vertex Cover, for instance, asks for a cover of size at most k, if one exists, or the answer NO, if not.

Vertex Cover (search version)

Input: A graph $G = (E, V)$, with vertex set V, edge set E, and positive integer value k.

Output: A vertex cover for G of size at most k. Otherwise output "no solution exists."

Independent Set (search version)

Input: A graph $G = (E, V)$, with vertex set V, and edge set E. An integer value k.

Output: An *independent set* for G, of size at least k, if one exists, otherwise output "no solution exists".

[5]A more informal way to present this problem is to state that if the graph represents a social network where the vertices are people and the edges are relationships between people, then the goal is to find whether or not there exists a group of at least a given number of people who did not know each other.

Euclidean Traveling Salesperson Problem (E-TSP) (search version)

Input: A finite set of points in the plane, $P = \{p_1, p_2, ..., p_n\}$, where the a distance, $d(p_i, p_j) \in R^+$ for each city-pair $p_i, p_j \in P$ is the Euclidean distance between p_i, p_j, and a bound, $B \in R$.

Output: A tour of all points in P with length at most B. In other words, an ordering $< p_{\pi(i)}, p_{\pi(2)}, ..., p_{\pi(n)} >$ of C, such that $\sum_{i=1}^{n-1}(d(p_{\pi(i)}, p_{\pi(i+1)})) + d(p_{\pi(n)}, p_{\pi(1)}) \leq B$ if one exists, otherwise output "no solution exists."

Computational complexity results are typically based on the decision version of problems; however it is not uncommon to see claims that the optimization versions of TSP (and variants) are *NP-complete*. This is not strictly true; however the optimization or search version of many problems can be converted (in polynomial time in terms of the input size) to the decision version, in which case the complexity of the search or optimization versions can be thought of as equivalent to that of the decision version. For example, if the optimization version of Vertex Cover is easy, then it could be used to solve the decision version of Vertex Cover in polynomial time, which would contradict the result that the decision version of Vertex Cover is *NP-complete*.

Here is an example of how this conversion could take place. Assume that the optimization version of Vertex Cover is easy. Given an instance, I, of Vertex Cover, solve it using our (fast) algorithm for the optimization version of Vertex Cover. Take the size of the solution, k', noting that k' is the smallest possible vertex cover for I. This value can be used to solve the decision version of the k-Vertex Cover, that asks if there exists a vertex cover of size at most k. If $k' \leq k$, then the solution is YES; otherwise the solution is NO, and therefore the decision version is easy.

Computational problems are free of unknowns and uncertainty and are therefore good candidates as instruments to better understand problem solving, because it is possible to understand the problem very well at the computational level. As stated earlier, however, this depends upon the assumption that the computational problem given, is in fact the problem that is being *solved* by the problem solver. We now identify why this might not be the case for the problems that have been the focus of study to date.

2.3.4 Can Hard Optimization Problems be Well-defined?

Computational problems, like those introduced earlier, can be classified as being in the class *P*, *NP*, or *NP-complete*, and are well-defined, at least as far as computer scientists

are concerned. They can be unambiguously defined, and, if they can be shown to be in the class *NP*, have solutions whose veracity can be checked in polynomial time. This would seem to imply that they are *well-defined* as far as human problem solving is concerned as well. However, in order to be well-defined in this sense, the goal must be verifiable; that is, the problem solver must be able to determine when the goal state has been achieved. This, as will be shown, may not in fact be the case.

Determining the optimality of a solution to E-TSP poses two main challenges. It requires, given a candidate solution, determining that it is optimal, which implies that no shorter tour exists. The infeasibility of this task is supported by results [72] that indicate that participants were not able to consistently identify the optimality of tours ranging between optimal (0% above optimal) and relatively sub-optimal (35% above optimal). More recent work has suggested that subjects may be able to differentiate between the relative difficulty of a set of instances of E-TSP; however, it is not immediately clear how this might relate to their ability to discern when a solution is optimal [26]. A second issue arises due to the Euclidean nature of the problem. Visually comparing the lengths of two tours may contribute to the immeasurability of the goal. Given two paths that are close in the length, it is possible that without the assistance of a cognitive aid, the problem solver may not be able to accurately determine which path is longer. This is true even if the issue of precision associated with measuring Euclidean distances is ignored. The human perceptual system is susceptible to errors when comparing relative lengths of lines due to the location and placement of other lines, as in the Ponzo illusion [73]. Participants' ability to find optimal solutions appears to also be sensitive to Gestalt principles of continuation or good form [118], as careful instance selection can result in highly sub-optimal tours [106]. Considering these factors, it is not likely feasible, in general, for problem solvers to measurably determine when a candidate solution is optimal; therefore the goal state is ill-defined.

In the n-Puzzle study conducted by [76], participants were tasked with finding the shortest sequence of tile movements needed to arrange the tiles in the given instances in numerical order. In order to determine whether a goal state has been achieved, the problem solver must determine a sequence of tile movements that will order the tiles such that the length of the sequence is minimized. The task of determining a sequence of movements resulting in all tiles being ordered appears feasible, as demonstrated by participants' ability to correctly arrange the tiles in numerical order [76]. Large instances of this problem may push the limits of attention (as even relatively small

instances required hundreds of moves to arrange the tiles in order); however, for the instance size used in the study ($7 - 15$ tiles) this does not appear to be an issue. The feasibility of this aspect of the goal is a byproduct of the fact that each move can be considered individually and working memory loads are therefore very low [9]. Determining the optimality of the path length, however, may require calculating the distance from an intermediate step to an optimal solution, or determining, given a sequence of moves that orders the tiles, that no shorter sequence exists. The likelihood of the first option is not supported by observations that participants are unable to even estimate the distance to a goal state with any certainty [76]. The second option is not feasibly calculable, due to the large number of possible sequences that must be considered. Known algorithms for determining optimal solutions to the 15-puzzle variant visited between $540,000$ and 6 billion states before finding optimal paths of length $41 - 66$ states [76, 82]. While it may be that some other method might be able to determine the optimality of a solution in fewer states, it is not clear what this method might be, and given the sub-optimality of participants' solutions, it is not likely identifiable by problem solvers either. Therefore, determining a given sequence is optimal in the n-Puzzle problem is infeasible and as a result the goal state of the problem is ill-defined.

Determining that a given vertex cover for a particular graph is optimal requires, given a candidate solution of size k, determining with certainty that there does not exist a valid solution of smaller size. If even only a small fraction of other candidate covers are considered to answer this question, this is well beyond the limits of working memory for all but trivial instances of the problem. For non-trivial instances, even comparing only a small number of covers quickly exceeds the limits of working memory [9]. It is possible that there exist other means of determining that a candidate cover is optimal for given instances, but it is not immediately clear what they may be. Again, since participants often fail to find optimal solutions, it is unlikely that such an alternative is readily available to novice problem solvers. Therefore, it is assumed that the task of determining the optimality of a candidate vertex cover is infeasible, and a goal state for this problem is ill-defined. Similarly, determining that a valid independent set is optimal amounts to showing that no larger independent set exists, a task that is infeasible, as the goal state for this problem definition is ill-defined.

Some aspect of a goal state of each of these optimization problems has been shown to be infeasible to identify, in general. Despite this, problem solvers, when presented an optimization problem, generate an internal representation of the problem that

allows them to determine when a goal state has been reached. It is therefore key to identify what goal(s) might be encoded within the internal representation, as this defines what problem is being solved. This in turn determines a sound base from which algorithmic explanations of human performance can be proposed.

Consider, again, the problem solver's internal representation of the problem. It consists of a problem space, defined by the start state, legal operators, and goal state of the problem. For many of the hard optimization problems studied to date, the start state and legal operators are well-defined, and therefore is it possible that they are consistent with those in the internal representation. Identification of a goal state, on the other hand, has been shown to be infeasible and cannot be encoded as such in the problem space. In studies of human performance on hard optimization problems, participants identify solutions to most (if not all) instances presented. These solutions, often sub-optimal, are selected because they satisfy some requirement or quality. Some solutions may also be the result of the problem solver giving up on the task of finding a state that matches a goal state, but this does not exclude the existence of a feasibly identifiable goal state. Sub-optimal performance results on hard optimization problems have typically been explained by proposing that problem solvers use approximations or heuristics [24, 25, 32, 52, 53, 56, 77]. However, this approach ignores the underlying issue of what problem is being solved; it ignores how the problem represented internally determines when a solution matches the goal. According to Newell and Simons information processing account [66], the problem solver determines that the instance of the problem is solved when a goal state is achieved, which is not possible if it is assumed that the original (infeasible to identify) goal state is encoded in the internal representation.

2.4 Summary

Many different kinds of problem exist that might be suitable for the study of human problem solving. Problems of transformation, free of uncertainty and unknowns, have proven useful for investigating how problem solvers navigate through problem spaces guaranteed to contain a path to the goal. In other words, they have made it possible to study performance on problems that are subjectively difficult, but solvable. Work in this area on problems whose problem spaces are relatively small has yielded interesting insights about factors that contribute to the subjective difficulty of these problems. It is also important to study how people cope with problems that, while

still solvable, are more complex, problems that are objectively difficult. Hard optimization problems seemed to offer such a possibility, as they are very closely related to computational problems for which no efficient algorithm exists. Rather surprisingly, human performance on instances of these objectively difficult tasks was found to be very good, with problem solvers finding near-optimal solutions very quickly, in many cases. However, meaningful interpretation of these performance results hinges on the assumption that problem solvers are indeed solving the problem given. If problem solvers are not able to encode part of these hard optimization tasks in their internal representation, then these performance results may not in fact be indicative of performance on the given task, but rather on some other, unknown, task.

The next chapter investigates this concept in more detail, by decomposing known hard optimization problems that have been used to study human performance, and attempting to determine if they are encodable by the human cognitive system.

Chapter 3

A Revised Approach to Studying Human Problem Solving of Hard Problems

Finding a solution to instances of the optimization version of some hard computational problems is likely beyond the power of the human cognitive system because the ill-defined goal is not encodable. If the given problem cannot be encoded then the problem solver cannot be solving the assigned task. While there are likely solvable instances of any non-encodable problem, the computational challenge of identifying the goal state renders countless other instances infeasible to solve. The seeming contradiction of using non-encodable problems as instruments to evaluate human problem solving does not preclude the possibility of continuing to use hard computational problems to gain an understanding of the power of the human cognitive system; however, it does imply that it may be appropriate to either modify the way these kinds of problem are presented or modify how performance results are evaluated. Two possible improvements to the existing problem-solving study design are identified. The first is to continue using hard optimization problems as instruments and based on the observation that these problems are not encodable, attempt to identify what problem(s) might be encoded before analyzing performance results. The second is to identify hard computational problems which are encodable by the human cognitive system and use them to evaluate human performance on hard computational problems. There are benefits to both approaches.

Hard problems, in particular those for which it may not be possible to determine

if a solution is correct, are everyday occurrences of the environment in which the human cognitive system has evolved. As a result, it would be surprising if the human cognitive system had not developed fast and efficient ways to cope with hard problem-solving situations. Using problems which are not necessarily encodable by the human cognitive system can allow for the investigation of possible mechanisms that the cognitive system can use to render the problems encodable.

On the other hand, hard problems which are theoretically encodable by the cognitive system, and are difficult due to their computational complexity alone, can be used as tools to investigate how problem solvers navigate problem spaces in which they are unable to easily determine a path to the goal. For even very hard problems, there can exist rules or strategies which can be used to reduce the size of the problem space of some instances. Encodable hard problems could allow us to observe whether or not problem solvers are able to identify, learn, and apply these strategies on later instances.

Previous work on problems of transformation proposed a number of general problem-solving strategies, like hill climbing, brute force, and means-ends analysis, as possible explanations of human performance on problems of transformation. While identifying such general problem-solving strategies is important, a different approach is considered here. This is in part due to the fact that it is unlikely that a general problem-solving strategy can be sufficient to find solutions to instances of problems which are shown to be computationally hard. Instead, people may acquire and apply strategies specific to classes of instances, and possibly acquire associated schemata. These strategies and schemata are introduced in Section 3.2.1 below, and all strategies and schemata identified for a problem are equally applicable to the optimization and search versions of that problem.

In order for a schema to be acquired, the problem solver must learn to associate a particular strategy with a property of the problem or instance. It is possible for strategies to be learned without an associated schema, amounting to having a tool without necessarily knowing exactly when it is best used. Associating success at solving an instance of a problem with a strategy could result in acquiring an associated schema, or conversely, associating failure with a strategy could prevent schema acquisition.

To date, studies of human performance on hard computational problems has focused on minimization problems, and little effort has been made to investigate human performance on hard maximization problems. The only maximization problem that

has been investigated to date is the Knapsack problem, performance on which was compared to the Vertex Cover problem [37]. While performance on the Knapsack problem in this study was found to be relatively less poor as instance size grew, than on the Vertex Cover problem, these problems differed in terms of representation and the instances used, and therefore the source of these differences is not immediately clear. It is, therefore, an open question whether there might be performance differences between minimization and maximization problems, independent of differences in representation and input.

In this work, two hard computational problems have been chosen to investigate the factors that impact performance on hard computational problems. The Vertex Cover and Independent Set problems were selected for a number of reasons. As not-Euclidean problems, they might not be vulnerable to the visual perceptual processes that have been found to impact performance on the E-TSP. They are also a natural choice for comparing performance between minimization and maximization problems. These two problems are very closely related as a solution to an instance of one of these problems will directly yield a solution to the other problem on the same input.

3.1 A Framework for Handling the Ill-Defined Goal of Hard Optimization Problems

Human solutions to instances of hard optimization problems can be close to optimal. The underlying assumption reflected in the language used to talk about performance results on these studies is that participants are solving the given task. This may seem to be a valid assumption, as participants typically demonstrate understanding of the task and do not verbalize that they are trying to solve a problem other than the one given. This, however, is likely a reflection of the adaptivity of our cognitive system. On the surface, this seems to be what Gigerenzer is referring to with *fast and frugal heuristics*: if a problem cannot be solved exactly then heuristics are a natural alternative. However, before assuming that heuristics or some other suboptimal strategies are the only way to explain performance, it is important to determine what problem might be encoded by the the problem solver because the encoded problem drives performance.

When tasked with instances of a hard optimization problems that are not encodable by the cognitive system, the problem solver encodes *some* problem, as indicated

by their ability to find a solution. These solutions can be very close to optimal, which implies that the encoded problem likely shares a great deal with the original unencodable problem. A method is proposed to identify alternative problem formulations which are encodable by the human cognitive system, based on breaking the problem down and modifying the starting problem as little as possible. Identifying candidate problems that can be encoded by the cognitive system makes it possible to propose models that might predict performance.

3.1.1 Problem Decomposition

Once a problem has been shown to have a goal that is infeasible to identify, one can try to identify alternative and feasible candidate goals that could be included in the internal representation. These modified goals can help to predict performance, which can be compared, experimentally, to human performance. Results of this comparison can serve to support or eliminate candidate problems defined by a modified goal.

Identification of candidate goal modification(s) can begin with decomposing the given problem's goal. Participants' ability to generate solutions that are valid, at least in terms of some aspects of the given problem's goal, indicates that some aspect of the original goal likely remains unmodified. Ideally only those aspects that are infeasible to identify need be modified. It is therefore prudent to divide the goal into those aspects that are feasibly identifiable and those that are not.

Consider the minimum Vertex Cover problem, whose goal it is to find a vertex cover of minimum size. Recall that given a graph $G = (E, V)$ as input, a valid vertex cover for G is a set of vertices U, subset of V, such that every edge in E is incident to at least one vertex in U. The optimization version of this problem asks for the smallest vertex cover. We can decompose this goal into two components, one that is feasible to identify, and another that is infeasible to identify. It is feasible to determine whether a given candidate vertex cover U is valid. For this, one can consider each edge in G independently, and check whether it is incident to at least one vertex in U. Each check in this process is independent and the load on working memory is low. Therefore, determining whether a set of vertices is a vertex cover is feasible. Determining whether a candidate cover is smallest, however, is in general not feasible, as discussed earlier. Candidate decompositions for some other problems are presented in Table 3.1, and explored in detail below.

From this decomposition, the aspect of the goal that is infeasible to identify can

Table 3.1: Decomposition of goal elements for select hard optimization problems. For each problem the goal is decomposed into those aspects which are feasible to encode, and those which are not.

Problem	**Feasible**	**Infeasible**
E-TSP	A valid tour	Length of tour minimized
n-Puzzle	Sequence to order tiles	Length of sequence minimized
Minimum Vertex Cover	A valid vertex cover	Size of vertex cover minimized

be clearly identified and potentially modified into one that is feasible to identify. It is important to note that most modifications of the goal for a problem will result in a different problem. This is true for all modifications proposed below. For any problem, there may be many possible ways such a modification can take place. An idealist approach is to assume that the goal is modified as parsimoniously as possible while retaining as much of the original goal structure as possible. This is in line with the idea that only infeasible to identify aspects are modified. It is possible that goal modifications do not adhere to this idealist perspective; however, it is a prudent starting point. Regardless of the level of modification, three distinct categories of goal modification are identified: restructuring modifications, general modifications, and specific modifications. Two problems, Vertex Cover and Independent Set, are analyzed in terms of these goal modifications.

3.1.2 Goal Modification Types

Restructuring Modifications

It may be possible for goal aspects, both feasible and infeasible, to be recombined in such a way that the goal is rendered feasible to identify. Local optimization is an example of such a restructuring mechanism, where the global optimization requirement is reduced to a local optimization, by shifting the target of optimization. This can be accomplished by recombining goal aspects in such a way that it is feasible to evaluate the new goal given the constraints of the cognitive system.

General Modification

A general modification is one that replaces an aspect of the goal with a general quality that can be feasibly evaluated, without including an exact value or measure. In this

case, the infeasible aspect of the goal is either eliminated or replaced with one that can be feasibly evaluated.

Specific Goal Modification

A specific goal modification is one in which an unidentifiable aspect of the goal is replaced with one or more quantitative measure or value. Such values can be refined and/or adjusted depending on whether or not a solution that satisfies the (previously) selected goal value can be found, resulting in an iterative strategy to solving the problem. The problem solver might estimate a value and attempt to find a solution that satisfies that value. If successful, the problem may either be deemed solved or a closer-to-optimal value may be chosen and the process might continue. If it fails, then the problem solver may choose a further-from-optimal value, and the process can continue. This modification results in an internal representation that is very close to the original optimization problem, except that the problem solver explicitly encodes the process of optimization by iteratively attempting to find solutions that are progressively closer to optimal. This would likely manifest in a goal searching mechanism that would involve a great deal of backtracking as a result of searching for a solution that matches the initial (and possibly subsequent) goal.

3.1.3 Goal Modification in Vertex Cover and Independent Set Problems

In this section the evaluation of possible general, restructuring, and specific goal modifications are illustrated, by decomposing the goal requirements for both Vertex Cover and Independent Set problems; candidate modifications are presented based on these decompositions; and predictions for performance based on these modifications. An overview of goal decompositions is given in Table 3.1, above. For an overview of predicted performance for each suggested goal modification, see Tables 3.2 and 3.3, below.

Minimum Vertex Cover

The infeasible aspect of the goal in the Minimum Vertex Cover problem is not that of finding a vertex cover (a vertex cover can be easily found by simply picking one vertex for each edge), but in determining the optimality of the vertex cover.

The goal can be restructured such that local minimum vertex covers are found in sub-regions of the graph, manifesting as a kind of local optimization. If the sub-graphs are small enough, then exhaustive search for a vertex cover becomes feasible, rendering the goal feasible to encode. This restructuring, while feasible, is challenging, as there may not be clearly identifiable sub-regions (or subgraphs) for a given graph. Further, sub-regions may not be mathematically well-defined by participants. Some instances may contain dense clusters (particular sub-regions) that are only loosely connected to each other. These would denote, visually, clear boundaries of subgraphs. On the other hand, graphs with relatively even density of edges may be more difficult to consistently decompose into sub-regions. As a result, in general, it may be difficult to maintain a mental model of how an instance is divided into sub-regions. A second consequence is that this division into subgraphs may not be static, changing during the problem-solving process. Identifying whether and when new subgraphs might be selected could be difficult. This modification predicts sub-optimal vertex covers, as locally optimal choices (in terms of subgraphs), may negatively impact the optimality of the overall solution. It also predicts some backtracking, as the problem solver attempts to find locally optimal covers for each subgraph or adjusts a minimum cover in one subgraph due to the impact of choices in a different subgraph.

A general modification to the infeasible aspect of the goal is to find a *minimal* vertex cover. A minimal vertex cover is a valid vertex cover that contains no redundancy that is, no single vertex can be removed from the cover without violating the requirements of a vertex cover. All optimal vertex covers are minimal however, not all minimal vertex covers are optimal. For instance, in Figure 3.1, a vertex cover comprising the coloured vertices is minimal, but not optimal. This property can be evaluated by examining each vertex in the cover individually, and determining whether its removal would render the cover invalid. The memory requirements for this evaluation are small, as each choice is local and it is dependent only on the vertices adjacent to, and edges incident to, that vertex. This modification would predict that upon finding a minimal cover, participants would not attempt to improve upon this solution, having matched this goal. This modification predicts participants finding fewer minimum vertex covers than if they are working on the given optimization task. It also predicts little or no backtracking once a minimal cover is found. To see why this is the case, consider the case where the problem solver searches for a minimal cover, which can be achieved by making local choices, for instance, finding a vertex that is not yet included in the cover that covers an uncovered edge. This strategy alone

is sufficient for finding a minimal cover and requires no backtracking. Finding an optimal cover however, may require trying to improve upon a minimal cover, which necessitates backtracking and trying different options from some earlier choice. An alternate general modification scheme involves discarding the optimization aspect of the goal entirely and simply searching for any vertex cover. This modification predicts suboptimal solutions, with little or no backtracking. However, it is not as desirable a modification as the previous one, as it less parsimoniously modifies the aspects of the goal by discarding any sense of optimization completely.

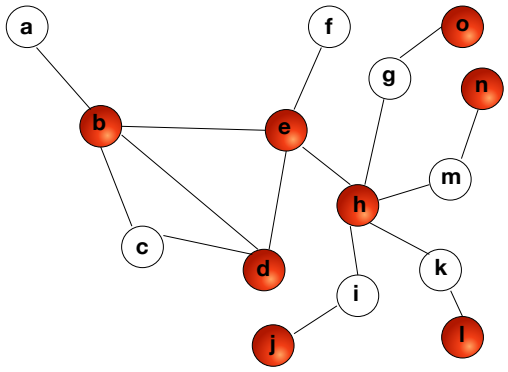


Figure 3.1: A minimal, and non-optimal, vertex cover. Vertices coloured red constitute a valid vertex cover.

Problem solvers may alternatively resort to a guess and check method to generate a feasibly measurable goal, and as a result they might generate a specific goal modification. This method seems especially appealing in problems with integer value measures, like Vertex Cover or Independent Set. Given an instance, a specific value k could be chosen by the participant, who could then search for a vertex cover no larger than k. If one is found, the solution might be deemed satisfactory or the value could be modified and another (better) solution could be searched for. Similarly if a solution is not found, the problem solver could either give up, or choose a further from optimal value and reiterate the process. This modification predicts that the amount of backtracking will be related to the optimality of the solutions. That is, more backtracking will be required if close to optimal specific values are chosen, and less backtracking will be required to find solutions that are not close to optimal. This modification results in a process of iteratively solving the search version of the problem for different values of k.

Table 3.2: Predicted performance for suggested goal modifications for the Minimum Vertex Cover problem. Predictions for each candidate modification are: the solution quality (relative to optimal), the number of optimal solutions found, and the amount of backtracking needed to find a solution.

Modification	**Solution Quality**	**Opt. Solutions**	**Backtracking**
Minimal VC	Sub-optimal, minimal	Few	Little to none
Any VC	Sub-opt., poss. non-minimal	Few	None
Local Optimization	Sub-opt., minimal	Few	Moderate amount
Guess and Check	Near-optimal	Many	Greater amount

Maximum Independent Set

Like Vertex Cover, the infeasible aspect of the goal in the maximum Independent Set problem is that of optimizing the size of the independent set. Due to the similarities these two problems share, the modifications are likewise similar.

The infeasible-to-evaluate goal could be restructured, resulting in searching for a maximal independent set. A maximal independent set is a valid independent set which cannot be made larger by simply adding another vertex to the set. This modification predicts that problem solvers would terminate search upon finding a maximal independent set, with little or no backtracking. An alternate general modification scheme is to find any independent set, such that the optimization goal aspect is discarded entirely. Due to the nature of the Independent Set problem, an empty set is an independent set, and therefore this modification is undesirable. It predicts potentially greatly suboptimal and even non-maximal solutions, with little or no backtracking.

The local optimization maximum Independent Set problem, where maximum independent sets are identified in subgraphs of the given instance, is also a viable candidate restructuring. It is vulnerable to the same issues as the local minimum Vertex Cover problem described above. It predicts suboptimal solutions, and even non-maximal solutions, with some backtracking.

Like minimum Vertex Cover, a specific goal modification is possible for this problem. Given an instance, a specific value k could be chosen and the problem solver could then search for an independent set with at least k vertices. If one is found, the solution might be deemed satisfactory or the value could be modified and another (better) solution could be searched for. If an independent set of size k is not found,

the problem solver could either give up or choose a further-from-optimal value and reiterate the process. Predictions for this guess-and-check modification are similar to that for Vertex Cover.

Table 3.3: Predicted performance for suggested goal modifications for maximum Independent Set problem. Predictions for each candidate modification are: the solution quality (relative to optimal), the number of optimal solutions found, and the amount of backtracking needed to find a solution.

Modification	**Solution Quality**	**Opt. Solutions**	**Backtracking**
Maximal IS	Sub-optimal, maximal	Few	Little to none
Any IS	Sub-opt., poss. non-maximal	Few	None
Local Optimization	Sub-opt., maximal	Few	Moderate amount
Guess and Check	Near-optimal	Many	Greater amount

In this section, candidate modifications have been identified for the optimization versions of the Vertex Cover and Independent Set problems. These proposed modifications, unlike the original optimization version of the problems, are encodable by the cognitive system. In both cases, the restructuring (minimal/maximal) and specific modifications are the most appealing because they maintain more of the original problem structure than the general modification, and also predict close to optimal solutions. Both these modifications predict close to optimal solutions, but differ in the amount of backtracking they predict. These predictions provide quantifiable measures for comparison to human performance results or computational model results.

3.2 Formulating Hard Problems with Well-Defined Goals

Alternatively, with the goal to study human performance on hard problems that are encodable by the human problem solver, well-defined problems, namely those with feasible to evaluate goals, can be used as instruments. Both the search and decision versions of NP-Complete problems, for instance, can be used as tools to investigate how problem solvers navigate problem spaces of computationally hard problems and what kinds of strategies may be identifiable by problem solvers.

3.2.1 Hard Search Problems

While both the search and decision versions of these problems are well defined, the search version has more in common with the optimization version, in the sense that it asks for a solution that meets the given constraints, whereas the decision version asks for an answer of YES or NO. So, while it is also interesting to consider human performance on the decision version, the search version is a more appropriate encodable alternative to the optimization versions of these two hard problems, as it allows more meaningful comparison of participant responses. In the following sections, we introduce predictions of human performance on the search version of two NP-Complete problems based on the potential to identify, learn, and apply strategies which may reduce the complexity of particular instances of these problems. The strategies identified are equally applicable to the optimization version of these problems, and therefore differences in performance between the optimization and search versions of these problems may be a result of 1) the differences in the encoded problem, or 2) differences in problem solvers ability to identify, learn or apply these strategies. This interaction may make it hard to tease apart these influences.

The number of moves needed to find solutions to instances of the search version of an NP-Complete problem, in general, is predicted to be significantly greater than the size of the solution. This follows from the complexity of the problem; unless the problem solver chances upon a valid solution, search for the goal will likely require backtracking from problem states from which the goal is not directly achievable. However, despite the fact that there likely does not exist a general algorithm for these problems, there are many instances which can be solved exactly quickly and therefore these predictions can be refined based on the type of instance presented. This is particularly true if participants are tasked with multiple instances of the problem and are able to acquire schemata. In order to proceed, a number of strategies are identified that are deemed suitable for human problem solvers to identify, learn, and apply, and based on these strategies, classify instances of these two problems. These classifications allow further refinement of predictions of performance results.

Strategies for the k-Vertex Cover Problem

The k-Vertex Cover problem is in the class FPT for parameter k, and a number of known *reduction rules* have been identified which, if applicable, could be applied to reduce an instance of Vertex Cover to a smaller kernel. Moreover, these reduction

rules can be applied in time polynomial to the size of the input, and therefore do not contribute to the *bad* time complexity of the problem.

Fixed parameter tractability has been proposed to explain the power of the human cognitive system [110], and therefore it is reasonable for to possibly explain the power of human problem solving. If the constraints of the power of the cognitive processes underlying human problem-solving ability can be explained by fixed parameter tractability, then it is possible that human problem solving can leverage fixed parameter tractability. A kernel has very specific meaning in fixed parameter tractability, and in the context of human problem solving, people may apply some but not all reduction rules and make a make a given instance easier, without reducing it to a true kernel. In this work, this will be referred to as finding a *reduced instance*. Three possible cases are identified; reduction rules may:

1. be sufficient to solve the instance exactly
2. reduce the instance to a reduced instance
3. not be applicable

It is natural to suppose that people may be able to leverage some reduction rules, acquiring associated schemata. Two known reduction rules are identified that, due to their simplicity, are especially well suited to the constraints of the cognitive system. Other reduction rules are known in the FPT community, but interest is limited to these two rules because they seem to be particularly appropriate for human problem solver.

The first reduction rule we define is the D1 strategy. Given an instance which contains one or more vertices of $degree - 1$ connected to a vertex v, it is never wrong to select v as part of a vertex cover, rather than its $degree - 1$ neighbours. The degree of a vertex is the number of vertices connected to it by an edge. In the graph in Figure 3.2, vertex D is such a vertex as it is connected to E, which is of $degree - 1$. The vertex D is can be added to the vertex cover and consequently the two incident edges are covered. While there are many instances which can be solved exactly by this strategy, there are many more instances for which it may not be sufficient to solve fully (case 2 or 3 above). Consider, for example, the instance in Figure 3.3, for which no $degree - 1$ vertices exist. The D1 strategy cannot be applied in this case.

A similar reduction rule, the D2 strategy is now defined. Given a triplet of vertices u, v, w that are connected by the edges $(u, v), (v, w)$ and (u, w), it is never suboptimal

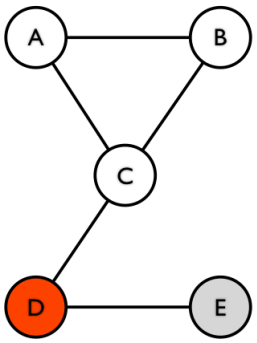


Figure 3.2: An undirected graph to which the D1 Strategy applies. Vertex D, coloured red, is a D1 candidate.

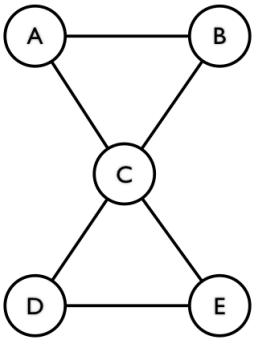


Figure 3.3: An undirected graph to which the D1 Strategy does not apply

to add at least two of these vertices to the vertex cover. Furthermore, if one of these vertices, say u, is of degree greater than two, then it is optimal to include u in the vertex cover. For example, in Figure 3.3, vertices A, B and C are such a triplet and vertices C and one of A or B can be added to the vertex cover.

Both the D2 and D1 rules are equally applicable to the optimization version of the Vertex Cover problem, as they do not depend on k and are a special case of a stronger rule. Although only these two special cases are considered, it is possible that under certain circumstances, people may be able to recognize other cases of this generalized rule. Given a graph G with vertices V, and edges E, for any subset U of V that forms a *clique*, at least $(|U| - 1)$ vertices must be in the minimum vertex cover. A clique is a set of k vertices where each vertex is connected by an edge to every other vertex in the clique. The D2 and D1 rules are special cases where $k = 3$ or $k = 2$, respectively.

A third reduction rule, which is not considered as a candidate for human problem solvers to identify or apply, is the k-Degree rule. Given an instance of the k-Vertex Cover problem, observe that if there exists a vertex, u of degree $m > k$, then if there exists a cover of size at most k, u must be in the cover. If not, in order to cover all m edges incident to u, all m neighbours of u must be in the cover; however, since $m > k$, then the cover size does not satisfy the requirements of the problem. It may be possible for human problem solvers to learn this reduction rule; however, it was not considered for this study. The rule's dependency on k, also renders this rule inapplicable for the optimization version of the problem.

Being able to identify and apply these strategies could significantly reduce the number of moves needed to find a solution by reducing the size of an instance to a smaller reduced instance. Further, should those instances be completely solvable by these strategies, it is predicted that participants who are able to identify and apply these strategies will find correct solutions more frequently. Successful application of these reduction rules can reduce the number of moves needed to find a solution, thereby decreasing the likelihood of giving up. For hard computational problems, the number of moves needed to find a solution can easily exceed the limits of the participants' attention and patience. This becomes less likely as the number of moves needed is reduced. In addition, each application of a reduction rule can be thought of as being independent of other subsequent moves, very much reducing the working memory load, as the resultant problem state is on a direct path to the goal. The acquisition and application of these moves can reduce the amount of backtracking

needed to find a path to the goal.

Strategies like these, based on reduction rules, will never result in adding a vertex to the cover that cannot belong to a valid solution; however, there exist many instances for which no known reduction rule applies and for which other strategies will be needed to solve. Another class of strategy is identified, a heuristic strategy, which may succeed for a subset of instances, and fail on others. Consider a commonly identified strategy for the k-Vertex Cover problem, namely that of adding high degree vertices to the vertex cover, often referred to as the *greedy* strategy. This strategy would be successful for the graph in Figure 3.4 as an instance of k-Vertex Cover for $k = 2$ and would result in the vertices D and C being identified correctly as a vertex cover of size 2. However, if applied to the graph in Figure 3.5 of the k-Vertex Cover problem for $k = 8$, this strategy would yield a cover of size 9 consisting of the vertices marked black, as opposed to the cover of size 8 indicated by the vertices marked blue. Unlike strategies that are correct in general for a problem, this type of strategy, as demonstrated, is not correct in general. To show that a strategy is a heuristic strategy it suffices to show that there exists an instance for which the strategy can be applied successfully and that there exists one for which it fails. The examples given above are proof that the greedy strategy is incorrect in general for the k-Vertex Cover Problem, and therefore is of a heuristic strategy.

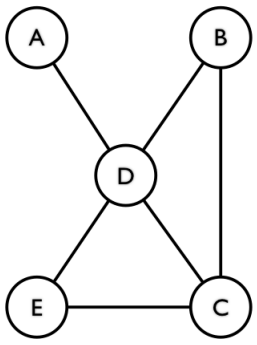


Figure 3.4: An undirected graph to which the Greedy Strategy succeeds. The application of the greedy strategy would result in vertex D, then C being added to the vertex cover.

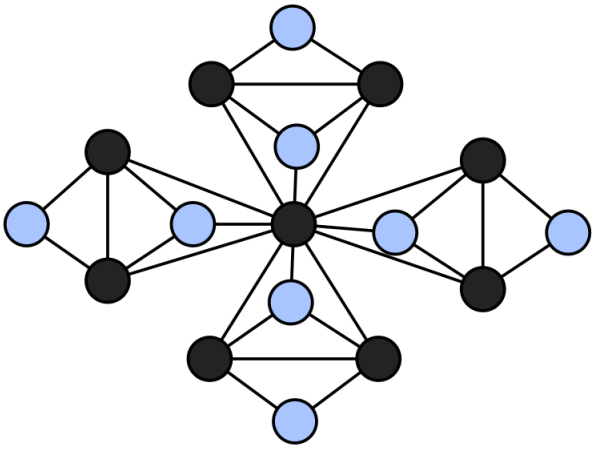


Figure 3.5: An undirected graph to which the Greedy Strategy fails. Blue coloured vertices constitute an optimal vertex cover of size 8, and black vertices constitute a sub-optimal vertex cover of size 9.

Strategies for the k-Independent Set Problem

Recall that solution to the Independent Set Problem can be derived directly from the solution of Vertex Cover problem on the same instance and that the k-Vertex Cover problem is in the class FPT. However, the Independent Set problem is not known to be in the class FPT. As a result, it is not likely that reduction rules can be used to convert any instance of the Independent Set problem to a reduced instance or kernel in such a way that the non-polynomial time complexity can be captured in terms of a small constant. Despite this, there exist reduction rules equivalent to the D1 and D2 rules identified for the Vertex Cover problem which can be applied to the Independent Set problem. This is because the two rules identified here are independent of the k value, of either the Independent Set problem or the associated solution to the Vertex Cover problem. So, while some instances can be reduced to a reduced instance or kernel, it is unlikely that there are reduction rules sufficient to do so such that the exponential time complexity can be limited to a small constant.

The D1 strategy for the Independent Set problem is defined as follows: given a graph G, if there exists a vertex $v1$ of degree 1, connected by an edge of degree 2, it is never suboptimal to include $v1$ in the independent set. The D2 strategy for the Independent Set problem is defined as follows: given a graph G containing a clique of size 3 containing at least one vertex of degree 3, and on vertex, $v2$, of degree exactly 2, then it is never suboptimal to include $v2$ in the independent set.

The heuristic Greedy strategy simply consists of adding low degree vertices to the cover.

Instance Classification

Instances of the Vertex Cover and Independent Set problems can be divided up into classes based on the applicability and success of these identified strategies. Instances of Vertex Cover, like that in Figure 3.5 for instance, can be classified as *greedy resistant*, because the greedy strategy applies (there are vertices of relatively higher degree), but if applied it will yield a sub-optimal solution. Instances of Vertex Cover, like that in Figure 3.2 for which the D1 strategy applies, can be classified on this basis. Similarly a class of problems is identified which are susceptible to the D2 strategy. In this context, class of instances can be identified, those to which neither the D1 or D2 strategy apply, and which the greedy strategy may (but not necessarily) fail.

Impact of Strategy Acquisition

Because strategies may not be applicable to all instances or may even, as in the case of heuristic strategies, fail to yield solutions to some instances, adopting a strategy for a problem can have either a positive or negative impact on performance. The nature of this impact depends both on the type of strategy learned and what strategies are effective on later instances.

Learning and applying either a Reduction Rule or Heuristic strategy can result in a positive impact on performance. Given a series of instances solvable by strategies of either type, the strategy can be learned and applied successfully on subsequent instances. This should result in improved performance in the form of higher incidence of correct solutions than if a general strategy is used instead. Successful strategy application should also reduce solution time, although lacking an associated schema this benefit may be small. Strategy application alone may still lead to a less-than-direct path to a solution, in particular if more than one strategy is known and tried. Reduction Rule strategy acquisition, given enough positive feedback, could result in an associated schema being acquired if the problem solver is able to identify a feature of problem states for which the strategy is correct. In terms of percent correct, schema activation would have a similar impact as strategy application, possibly resulting in a greater increase in correct solutions. It should also result in a more pronounced reduction in solution time, as it would eliminate the need for search at problem states

at which the schema is activated.

A negative impact on performance can result from learning a Heuristic strategy, and then encountering instances that are resistant to that strategy, if no alternate strategy is known or learned. Applying the Heuristic strategy to instances that are not resistant to it, as described above, could result in improved correctness, and if schema activation occurs, faster solution times. However, being then tasked with instances resistant to this learned Heuristic strategy could result in longer solution times, as either a general search is performed, or alternate strategies are applied. A decrease in the number of correctly solved instances is also predicted. Learning to reject a Heuristic strategy could also result in poorer performance on instances which are susceptible to the strategy, if its failure on previous instances decreases the likelihood with which it is applied. However, this decrease in performance may not be observed if previous success with it outweighs its failure on a few instances. Switching from instances to which a Reduction Rule applies to those to which it does not apply could have varying impact, dependent on whether or not the applicable strategy is used, and whether or not other strategies are also applicable to these latter instances.

Schema activation can also have either a positive or negative impact on performance on later instances. Correctly learning and applying a strategy can also lead to acquiring a schema associated with this strategy and result in a higher number of correct solutions and a reduced solution time. Schema activation at or near the start state of an instance of a problem should reduce the time needed to find a solution, if applied correctly, as no search is needed to find a solution. Schema activation later in the problem-solving process should still reduce solution time, if applied correctly, but perhaps less perceptively. In both cases, if the schema activated is correct, an increase in the percentage of correct solutions, relative to no schema activation, should result. Heuristic schema activation for an instance resistant to the associated strategy could result in an increase in solution time, relative to instances for which the strategy was correct. Failure would require either searching for a different strategy, or general search, both of which are slower processes than solution by schema activation.

More than one strategy may be learned for a given set of instances, and the interaction between competing strategies may be challenging to interpret. Further, some strategies are likely easier to learn than others. The above tables are meant to predict performance in the idealistic conditions where only one strategy is available to learn in a series of instances of a problem, and is unlikely to capture the complexity of how competing strategy acquisition and application can impact performance. How-

ever, it is still important to consider how a single strategy's acquisition can impact performance given instances to which it may or may not be applicable or correct.

3.3 Research Hypotheses

Previous work investigating human problem solving on hard optimization problems and problems of transformation have identified a number of factors that can impact performance. In this work, additional factors have been identified which are predicted to impact performance. These factors include how feasible the goal is to encode, minimization versus maximization, whether or not correct strategies can be successfully applied, and whether or not heuristic strategies can be avoided if not applicable. While strategy acquisition and application have been studied previously in studies of human performance on problems of transformation, they have yet to be investigated on hard computational problems. Further, no distinction has been made in previous work between heuristic and correct strategies. Goal specificity has been investigated in previous work; however, these studies used problems of transformation which are not necessarily as objectively difficult and more importantly did not result in instances of problems with goals that were not encodable.

These predictions lead to the following research hypotheses, which will be evaluated in the next Chapter.

Hypothesis 1 (Impact of Non-Euclidean Input on Performance)**.** *Performance on not-Euclidean hard optimization problems will be similar to that seen on Euclidean hard optimization problems.*

Hypothesis 2 (Relative Performance on Quasi-Equivalent Maximization versus Minimization Problems)**.** *Performance on a minimization problem will be better than on a similar and related maximization problem because minimization is less taxing on working memory.*

Hypothesis 3 (Impact of Infeasible to Evaluate Goal on Performance)**.** *Performance on the search version of both the Vertex Cover and Independent Set problems will be better in terms of solution quality than on optimization version of the same problem. Due to the infeasible-to-evaluate goal of the Optimization version of these problems, problem solvers will not be solving the given task and as a result their performance, gauged in terms of relative performance on an encodable variant of the same problem, will deviate from the given goal.*

Hypothesis 4 (Goal Modification of Problems with Infeasible to Encode Goal and its Impact on Performance)**.** *An infeasible-to-encode and identify goal state is modified into a feasible-to-identify goal state. As a result, solutions to problems with infeasible-to-encode goal states will better match solutions to a related problem with a feasible-to-identify goal than to the given task.*

Hypothesis 5 (Impact of Goal Modification on Cognitive Load)**.** *Modification of an ill-defined problem* Π *into related well-defined problem* Π^1 *will result in* Π^1 *being easier, and therefore having a lower cognitive load than* Π*. If the resultant problem is easier, then the number of moves needed to find a solution will be significantly fewer than on the given task. A reduced cognitive load could also manifest in less dependence on cognitive aids to solve the problem.*

Hypothesis 6 (Impact of Instance Type)**.** *Instance type can impact performance.*

Hypothesis 7 (Impact of Instance Type Ordering)**.** *Ordering of instance types can impact performance.*

Hypothesis 8 (Ability to Acquire Correct Strategies)**.** *Problem solvers are able to acquire and apply correct strategies, in particular those based on known reduction the rules: D1 and D2 strategies.*

Hypothesis 9 (Impact of Acquiring Correct Strategies)**.** *Instance type and order can impact performance in a positive manner. Interaction or experience with an instance class impacts which strategies can potentially be identified, learned, and applied on subsequent instances of the problem. The opportunity to learn Reduction rules earlier on may be associated with better performance on later instances. If an associated schema is activated this improved performance should be coupled with reduced search, and reduced cognitive load.*

Hypothesis 10 (Ability to Avoid the Application of Heuristic Strategies)**.** *Participants apply Greedy strategy. In the Vertex Cover condition they can learn to avoid it, upon encountering instances resistant to it, if able to determine the strategy has failed. This is more likely in the Search condition, due to feedback of missing the goal. In Optimization condition, modification may result in Greedy Resistant instances not being resistant to this strategy.*

Hypothesis 11 (Impact of Acquiring Heuristic Strategies)**.** *Instance type and order can impact performance in a negative manner. Interaction or experience with an instance class impacts which strategies can potentially be identified, learned, and applied on subsequent instances of the problem. The opportunity to learn a Heuristic strategy earlier on may be associated with poorer performance on later instances in particular those resistant to that Heuristic strategy.*

Chapter 4

Experiments

4.1 Goals of the Study

The two versions of the two different problems used in this study provide a unique opportunity to explore how performance differs on hard computational problems that are very closely related. The two problems are very closely related as the solution to one will always directly yield the solution to the other. The two versions of these problems, the Search and Optimization versions, as used in this experiment, share identical goal states; however, they differ in how the goals are presented to the problem solvers, and therefore they differ in the way in which they can be represented internally by problem solvers. This design provides a mechanism to explore previously unexplored scenarios, described in detail in the following sections.

4.1.1 General Performance

One of the original motivations of this study was to evaluate human performance on not-Euclidean hard computational problems in order to compare it to previous work on Euclidean hard computational problems, notably the Euclidean Travelling Salesperson Problem (See, e.g., [10, 24, 25, 32, 35, 36, 45, 52, 54, 55, 72, 77, 85, 106, 114, 116]). If similar human performance results are observed on not-Euclidean problems this would support the hypothesis that the reported high quality and fast timing results were a result of problem-solving mechanisms not wholly dependent on Gestalt visual processes, like good continuation or good form [118]. This does not discount the possibility that the visual system is not somehow being leveraged (as proposed in [77]) but rather would serve to discount Gestalt mechanisms as the sole

source of solution quality as these mechanisms do not directly apply to instances of not-Euclidean graph problems.

4.1.2 Maximizing vs. Minimizing

A second purpose of this work was to take a first look at how performance varies between minimization and maximization problems given problems with the same representation and input. There are many possible reasons for performance to differ between instances of two different problems, and therefore to minimize possible sources of differences two graph problems were chosen for this study. These two problems, the Vertex Cover and Independent Set problems, while likely differing in terms of their parameterized complexity, are still very closely related as the solution to one directly yields a solution to the other on any instance[1]. This property makes them an ideal pair of problems for this purpose; they constitute a maximization and minimization problem with the unique quality that, if the same instances are given, equivalent solutions and measures of solution quality can be easily derived using equivalent steps or moves.

One of the challenges of comparing performance on different problems, in particular when comparing solutions for maximization problems to those for minimization problems, is that the size of the solutions may differ. This is also an issue when comparing solution quality of different instances of the same problem. Common ways of converting solution quality measures into those which can be meaningfully compared are to either standardize or normalize the measure [112]. Normalization is computed based on the size of the optimal solution to an instance of a problem Π, S_Π, and can be defined as:

$$S_\Pi = \frac{S_{obs} - S_{opt}}{S_{opt}} \tag{4.1}$$

S_{obs} denotes the size of the observed solution and S_{opt} denotes the size of an optimal solution. This measure in turn can be converted to a percent above optimal. A problem arises when using this normalized measure to compare performance on maximization and minimization problems because the optimal solution sizes potentially confound the measure for minimization problems with small optimal solutions or conversely for maximization problems with large optimal solutions. This issue can

[1]When parameterized by the solution size to be determined Vertex Cover is known to be in the class FPT, whereas Independent Set is not.

be avoided for the two problems chosen for this work by converting an independent set solution into its associated vertex cover solution allowing PAOs to be compared without inflating or deflating the measure.

4.1.3 The Role of the Goal

Another purpose of this experiment was to evaluate how the identifiability of the goal impacts performance on hard computational problems by tasking participants with hard computational problems with goals that differed only in terms of how feasible they were to identify. Participants were tasked with either the Optimization or Search version of the Vertex Cover or Independent Set problem. For the Search version of each problem the size of the given goal was exactly that of an optimal solution; thus participants in both Problem Version conditions of the same problem were given equivalent goals except that in the Search condition the goal size was specified. The role of the feasibility of identifying the goal could thus be examined as the remainder of the problem definitions are equivalent between the Search and Optimization versions of each problem. If, as predicted, subjects modify the goal when given the Optimization version then this would manifest in differences between the Search and Optimization groups in terms of the number of optimal solutions, the quality of solution, and amount of backtracking.

4.1.4 Strategy Acquisition

A third purpose of this experiment was to investigate if participants would be able to identify and apply identified strategies when tasked with these problems. Two classes of strategies were identified: correct and heuristic.

Correct strategies, if identified acquired, and applied correctly, will never lead to suboptimal or incorrect solutions. Once they acquired there is no reason for these strategies to be discarded completely; however, this does not discount the possibility that even if, once they are acquired, other strategies might still be applied. Recall that acquiring a strategy does not guarantee its application only that it is one of possibly many strategies that the problem solver has available to them. Schema activation, on the other hand, means that a strategy will be applied when it is applicable. The effect of correct strategy acquisition is predicted to interact with the version of the problem given. In the Search version of either problem the application of a correct strategy which leads to an optimal solution to an instance can potentially reinforce the cor-

rectness of the strategy making it more likely to be applied later and possibly leading to schema acquisition. In the Optimization version, however, this reinforcement may not take place as there is no external feedback indicating that a solution is correct. However, if goal modification results in another problem being solved, the problem solver may infer the correctness of a strategy based on their ability to match the new goal. The effect of correct strategy acquisition is also predicted to interact with Graph Type. Graphs were purposely generated to be vulnerable to correct strategies. The application of the appropriate correct strategy should result in closer-to-optimal solutions, and more optimal solutions. Schema activation predicts a reduction in the amount of search needed.

Heuristic strategies, on the other hand, can lead to suboptimal solutions on instances to which they are resistant. If the problem solver is able to associate failure to find an optimal solution with the application of a heuristic strategy then the strategy may be avoided or rejected completely. To test this theory strategy-resistant instances were designed to not be solvable by that strategy. The effect of heuristic strategy acquisition is predicted to interact with the version of problem given. In the Search version of either problem, the application of heuristic strategies where they lead to a suboptimal solution could be more easily noticed if the size of the solution exceeds the given goal. As a result participants in the Search condition may be more likely to avoid heuristic strategies if they lead to solutions which do not match the given goal than participants in the Optimization condition. However, if goal modification in the Optimization condition results in a problem being solved other than the given task then the problem solver may infer the correctness of a strategy based on their ability to match the modified goal. The effect of heuristic strategy acquisition is also predicted to interact with the Graph Type. Graphs resistant to acquired strategies could be rendered more difficult until the subject identifies that the strategy is suboptimal. This could manifest as a greater number of moves to find a solution, fewer optimal solutions, solutions which are further from optimal, and increased cognitive load. Graphs vulnerable to acquired strategies could be less difficult, which could manifest as requiring few moves to solve, finding more optimal solutions, and finding solutions which are closer-to-optimal.

4.2 Pilot Studies

Pilot studies were performed prior to this research to investigate human performance on the Optimization version of the Vertex Cover and Independent Set problems. In these studies participants worked with paper and pencil versions of the problems, and as such, they were responsible for all bookkeeping tasks. These tasks included keeping track of previous vertex selections, determining which edges were covered (in the Vertex Cover condition), determining which vertices were adjacent to a selected vertex (in the Independent Set condition), and the size of the current solution. Cognitive aids were provided in the form of coloured pens to allow participants to use different coloured pens for different kinds of moves or selections; however, it was found that few participants used different colours to assist in their problem-solving process. Participants were observed, however, keeping track of which edges were covered in the Vertex Cover condition by crossing-out covered edges. These observations influenced the design of this study.

4.3 Methodology

This study and the methodology chosen for it was based in part on pilot studies of human performance on the Optimization version of both the Vertex Cover problem and the Independent Set problem, as described earlier. Participants were presented with instances of the appropriate problem on an iPad making it possible to provide cognitive support not previously provided in the pilot studies.

4.3.1 Participants

Ninety-six undergraduate students participated in the study at the University of Victoria for optional extra credit in a psychology course.

4.3.2 Materials

A set of 25 graphs was presented to all participants. The instrument included 10 small graphs with 10-19 vertices, 10 medium graphs with 20-29 vertices and 5 large graphs with 30-45 vertices. For a summary of select properties of all graphs see Table A.1. Images of all graphs are included in Appendix A.3. Five types of graph were generated with five graphs per type: D1, D2, Greedy Resistant (GR), No Strategy

(NS), and Random (Rnd). With the exception of the Random graphs all graphs were generated by hand (as described below) by the researcher and the members of the researcher's research group.

D1 (D2) graphs were generated so they included D1 (D2) vertices at the onset. D1 and D2 vertices, respectively, are vertices that would be selected by D1 and D2 strategies as described in Section 3.2.1 above. The number of D1 (D2) vertices included was kept small, relative to the number of vertices in the graph, to lower the likelihood of one being selected at random.

Greedy Resistant graphs were generated so that the inclusion of the highest degree vertices in the solution would result (for the Vertex Cover problem) in a sub-optimal solution. The highest degree vertices were those with either the highest degree or second highest degree overall for that instance and the number of high degree vertices varied based on the number of vertices in the instance. For a summary, see Table 4.1. To illustrate, a sub-optimal solution is shown for graph 14 in Figure 4.2, and an optimal solution is shown in Figure 4.1. A small number of the applicable vertices were included in order to lower the probability of their being selected at random.

Table 4.1: Properties of Greedy Resistant graphs. Each Greedy Resistant graph included a small number ($NumD_{high}$) of high degree D vertices, relative to the number of vertices (n) in the graph. A vertex was considered high degree if it was highest or second highest, compared to other vertices in the graph.

Graph ID	$NumD_{high}$	**n**	D
10	1	13	6
11	1	19	8
12	2	22	5, 5
13	2	25	5, 5
14	3	43	6, 6, 7

No Strategy graphs were generated such that they included no D1 or D2 vertices at the onset and were not resistant to the Greedy strategy.

Finally, Random graphs were generated as follows. A base graph was generated by hand by connecting all vertices in a spanning tree (and therefore with $e = n - 1$ edges). This was done to ensure that all randomly generated graphs were connected and did not include disconnected components. Edges were added to this spanning tree. The number of edges was determined by $e \times 0.75$ where e is the number of edges in the base graph. Each edge was added by choosing a start and end point at random

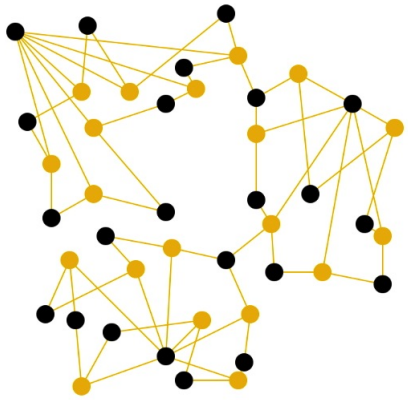


Figure 4.1: An optimal vertex cover of size 20 to Graph 14. Yellow vertices are in the cover. Yellow edges are covered by those vertices.

from the set of vertices in the graph. Duplicate edges, if generated, were discarded. Similarly edges with start point equal to end point were discarded. As a result, the number of edges in each of these randomly generated graphs was approximately $(n-1) + 0.75 \times (n-1)$ with some variation due to ignored edges.

Table 4.2: Five different block orderings. Participants were randomly assigned to one of these four block order groups.

Group	**Block 1**	**Block 2**	**Block 3**	**Block 4**	**Block 5**
A	D1	D2	GR	NS	Random
B	D2	NS	D1	GR	Random
C	GR	D1	NS	D2	Random
D	NS	GR	D2	D1	Random

Graphs were blocked by type with five graphs per block. Each block began and ended with a small graph chosen at random from the small graphs of that type. The remaining three graphs (two medium, one large) were randomly ordered between these two small graphs. The placement of small graphs at the start and end of each block was done to mask the transitions between blocks.

The order of blocks was determined as follows. Four different orders of D1, D2, Greedy Resistant (GR), and No Strategy (NS) blocks were designed so that between these four orderings no block always preceded another. The block of Random graphs was always presented last in order to maximize the chance that all purposely-

Figure 4.2: A sub-optimal vertex cover of size 23 to Graph 14. Yellow vertices are in the cover. Yellow edges are covered by those vertices.

generated graphs would be completed. See Table 4.2 for details.

4.3.3 Procedure

Participants were randomly assigned to conditions as follows. Half of participants were instructed to solve the Vertex Cover problem and the other half were instructed to solve the Independent Set problem. Within each of these conditions half of participants were assigned to the Optimization (OPT) condition and half were assigned to the Search (SRC) condition. See Table 4.3 for details. Participants were then randomly assigned to one of four block orderings (A, B, C, or D, as described in Table 4.2, above).

Table 4.3: Number of participants randomly assigned to each condition

	Vertex Cover	**Independent Set**	**Total**
Optimization Version	24	24	48
Search Version	24	24	48
Total	48	48	96

The Vertex Cover problem was presented as the Guard Problem. Participants were told that the graphs represented art galleries where the vertices were guard posts and

the edges were the hallways containing priceless art. Participants in the Optimization group were asked to find the fewest guards needed to cover all the hallways. See Figure A.1 in Appendix A.2 for the instructions given to participants. Participants in the Search group of the Vertex Cover problem were asked whether they could find a way to cover all the hallways with at most a given number of guards. If they deemed the task not possible they entered "NO" into the "goal-possible" field. See Figure A.2 in Appendix A.2 for participant instructions. For all graphs the value given was the size of a minimum vertex cover so that participants in both Vertex Cover conditions were tasked with the same goal.

The Independent Set problem was presented as the independent people problem. Participants were told that the graphs represented social networks and that the vertices were people and the edges were relationships between people. Participants in the Optimization group were asked to find the largest group of people who did not know each other, as shown in Figure A.3. Participants in the Search group of the Independent Set problem were asked whether they could find a group of at least a given number of people who did not know each other. If they deemed the task was not possible they were instructed to enter "NO" into the "goal possible" field. See Figure A.4 for participant instructions. In all instances the value given was the size of a maximum independent set so that all participants in both Independent Set conditions were tasked the same goal.

Cognitive Support

Based in part on observations of participant bookkeeping strategies in pilot studies a number of cognitive supports in the form of working-memory support was provided by the instrument in order to facilitate problem solving. The use of a digital touch interface facilitated the addition of these cognitive supports. Working-memory theory predicts that processing power is reduced when storage capacity is taxed [9]. In pilot studies using the Vertex Cover problem participants were observed *crossing off* edges when they were covered by a vertex. Visual support was provided in both problems to indicate which vertices were in the candidate solution. For the Vertex Cover problem when a vertex was added to the cover its colour was changed to indicate its inclusion, and all incident edges were likewise coloured. When a vertex was de-selected its colour was reverted to the default colour, and all incident edges not covered by another vertex were likewise reverted to the default colour. For the Independent Set problem when

a vertex was added to the independent set, its colour and that of all neighbouring vertices was changed indicating that they were not independent of each other. When a vertex was removed from the independent set, its colour and that of its neighbours was reverted to the default colour if the vertex in question was not the neighbour of another vertex in the independent set.

Participants were provided with a display of the size of their current solution thus eliminating the need to keep a tally of the size of the solution and minimizing the possibility of simple bookkeeping errors. This choice was made to support participants in their problem-solving task by freeing them from less interesting tasks, thereby reducing working-memory load.

An undo button was provided to assist in backtracking, which allowed participants to sequentially undo previous selections or deselections. Participants could also undo any move manually by deselecting any vertex currently selected.

When a participant deemed a solution complete the next instance could be reached by clicking a button labelled "Next", as long as the solution was valid. Participants in the Search condition could also give up and move to the next instance by first entering "NO" into the goal possible field before selecting the "Next" button. An independent set was considered valid if no adjacent vertices were included in the set[2]. A vertex cover was considered valid if every edge was covered by at least one vertex. These checks for validity were implemented to avoid participants mistakenly moving on to the next instance before satisfying the feasibly determinable goal aspect.

The interface as experienced by participants in the Search version of the Vertex Cover condition is shown in Figure 4.3.

Verbal Protocols

The use of verbal protocols was considered to gain a better understanding of the underlying problem-solving strategies used by problem solvers. Eight participants were given a subset of the instances used in the main study and were video and audio recorded while working on the instances of the problem. Participants were encouraged to verbally report their thought processes during this time. The results of these verbal protocols are not reported because they were not readily interpretable. After further research it became clear that this was because the interface used for this study was not properly designed to allow participants to meaningfully verbalize their actions. In

[2]A consequence of this is that an empty independent set is valid.

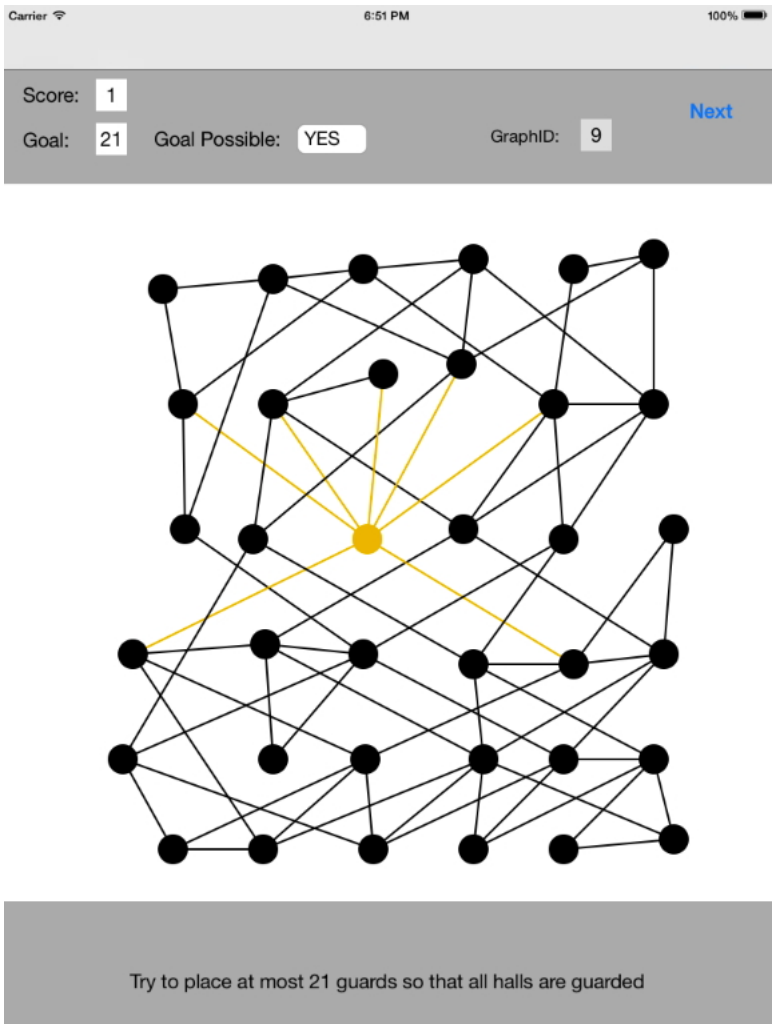


Figure 4.3: The iPad interface as presented to participants in the Search version of the Vertex Cover condition.

order for a thought to be verbalized it must have a verbal representation in working memory [66]. Because the objects being manipulated by participants on the iPad had no meaningful verbal representation that distinguished one from another (all vertices were identically sized and shaped and differed only in terms of location) participants' verbalization of thoughts amounted to "pick this one" or "that one". This rendered the verbal output difficult to interpret.

4.3.4 Analysis

Factors

The analysis considers a number of factors and their interactions, and a number of measures. The factors considered are:

- Problem (Independent Set versus Vertex Cover)
- Problem Version (Search versus Optimization)
- Graph Type (D1, D2, GR, NS, or Random)
- Group (instance order)
- Interactions between factors

The Problem factor compares performance between two different problems given the same graphs, as well as between a maximization and minimization problem. The Problem Version factor investigates differences in performance between the Search and Optimization versions, as well as proposed goal modifications. The interaction between Problem and Problem Version factors investigates how differences in performance between the Search and Optimization versions manifests between these two different problems. Graph Type investigates the relative difficulty of the different graph types. Interactions between Graph Type, Problem and Problem Version are considered to evaluate how instance difficulty differs between these factors. The Group factor investigates the impact of instance order, independent of Graph Type. The interaction between Graph Type and Group is considered to investigate the acquisition of the identified strategies leading to improved or poorer performance on later blocks.

Performance Measures

A number of performance measures are considered, including:

- % Optimal solutions
- PAO
- % Maximal/Minimal solutions

- Number of Moves
- Time per Move
- Number of Undos
- Number of Toggles

Solution quality was investigated using a number of different measures. In all conditions, the percent of optimal solutions was calculated. This value is appropriate for comparing performance between all conditions. In the Optimization condition of both problems, PAO was computed to compare the relative performance between problems, and also to compare performance on these not-Euclidean problems to previously studied hard computational problems. The percent of maximal (Independent Set problem) and minimal (Vertex Cover problem) solutions found was computed. While these measures are not necessarily an indication of correct or perfect solutions to the problems given they are a relative measure of performance.

It is common to use standardized or normalized measures when comparing performance on hard computational models between conditions (See, e.g., [54, 32, 115, 112]). In this work percent above optimal (PAO) is computed for the Optimization condition of both problems. For the vertex cover problem, the PAO is computed as:

$$S_{VC} = \frac{S_{obs} - S_{opt}}{S_{opt}} \tag{4.2}$$

For the Independent Set problem the equivalent measure is calculated by converting the size of the independent set found to the size of its associated vertex cover so that measures could be compared between the two conditions. From this value the associated PAO can be calculated.

PAO is not a valid measure for the Search condition in this experiment because a subject may decide that the given goal on an instance was not achievable and move on to the next instance. As a result, a skipped instance may not have a valid solution associated with it and therefore the size of the solution cannot be assumed to be valid. Instead the frequency of optimal solutions found over the 25 instances is used to compare performance between conditions.

Another measure of performance is the amount of search needed to find a solution. Both the number of Moves needed as well as the amount of backtracking (or Undos) needed to find a solution are presented.

An indirect measure of performance is the amount of cognitive load experienced by participants while problem solving. Two measures of cognitive load are considered: the Time per Move, and the number of Toggles. The amount of time spent making a move selection can be seen as a measure of the amount of internal effort that is needed to make a move. Therefore, the Time per Move is also evaluated. It was also observed that participants frequently toggled vertices on and off while trying to find a solution. While it is possible that these moves constituted legal additions or removal of vertices from the candidate solution, an alternate explanation is that they were making use of the user interface to test the impact of a move that was up for consideration. A move was considered a Toggle if the currently selected vertex was selected in the previous move. Toggles could be indication of increased cognitive load.

Measures of Strategy Acquisition and Application

In addition to performance measures, other measures are considered to investigate strategy acquisition and application, including:

- Degree of selections
- Proximity of subsequent selections (Path Length and Euclidean Distance)
- D1 selections
- D2 selections

The mean Degree of each selection is presented to evaluate whether or not moves might be driven by the Greedy strategy. Two measures of local selections, Path Length and Euclidean Distance, are presented to investigate if a local strategy or local optimization is at play in the move-selection process. Finally, the number of D1 and D2 selections are presented.

Analysis Tools and Data Collection

Bayes factors derived from standard analysis of variance were used to investigate evidence for or against the above-stated factors [68, 83] for each of the listed measures. Bayes factor analysis was adopted for this analysis instead of null hypothesis significance testing (NHST) for a number of reasons. NHST provides a way to evaluate the probability of the observed outcome (the data) over null hypothesis [68]. As a result, is not possible using NHST to accept the null hypothesis. In scientific study

it is typical to want to evaluate the probability of a hypothesis given the observed outcome. Bayes factor analysis, on the other hand, based on Bayes' theorem shown in Equation 4.3, provides a way to accept or reject either of two models, based on the relative posterior probabilities of competing models.

$$p(H|D) = \frac{p(D|H) \cdot p(H)}{p(D)} \tag{4.3}$$

Given two competing hypotheses, H_1 and H_2, the Bayes factor is computed to evaluate a change in prior odds based on the observed data. The Bayes factor $\frac{p(D|H_1)}{p(D|H_2)}$ is computed from the posterior odds $\frac{p(H_1|D)}{p(H_2|D)}$ and prior odds $\frac{p(H_1)}{p(H_2)}$ of each hypothesis according to Equation 4.4. Evidence for model H_1 against H_2 is defined according to [80] (p. 139), with a Bayes factor of $1-3$ indicating weak evidence, $3-20$ indicating positive evidence, $20-150$ indicating strong evidence, and >150 indicating very strong evidence.

$$\frac{p(H_1|D)}{p(H_2|D)} = \frac{p(D|H_1)}{p(D|H_2)} \cdot \frac{p(H_1)}{p(H_2)} \tag{4.4}$$

To evaluate the impact of each of the factors considered in this work, Bayes factors for all models created by removing or leaving all main effects or their interactions from the full model were computed. JZS priors were used in the calculations, as suggested in [83]. All analysis was conducted using R [1], and Bayes factors were computed using anovaBF in the R package developed by [64]. Pearson's r was used to investigate possible correlation between some measures.

The instrument used by participants generated a log in the background containing data for each instance, including whether or not the solution was valid, the size of the participant's solution, the start and end times (in milliseconds), the number of Undos, and a list of all vertices touched. Perl scripts were used to convert this data into .csv (comma separated value) files, and to compute a number of quantitative values for each solution, including the number of D1 and D2 selections made, solution size, $\Delta - Opt$, distance (path and Euclidean) between subsequent solutions, number of vertices touched, and the degree. The $\Delta - Opt$ was calculated by subtracting the optimal solution size from the subject's solution size and taking its absolute value. Scripts were written to compute the number of valid solutions and the number of maximal or minimal solutions. This was done to minimize human error during the processing and analysis of participant data. Values were computed from this data using spreadsheet equations including normalized and standardized values, PAO, and

differences.

4.4 Results

For clarity results in this section are broken down by the associated research hypothesis being addressed. A great number of performance measures are considered and analyzed in terms of a number of factors and their interactions. For clarity only those results with positive or stronger evidence are discussed in detail and the analysis is presented later in Chapter 5. Details of all measures are found in the Appendix in Section A.4.2 and details of evidence for all factors and their interactions are found in Tables 4.5 and 4.12.

Table 4.4: Summary of performance results for all conditions. Mean values presented.

	Independent Set		**Vertex Cover**	
	Search	**Optimization**	**Search**	**Optimization**
% Optimal	83.75	57.23	70.13	40.67
PAO	NA	10.74	NA	9.60
% Maximal/Minimal	95.00	91.00	96.00	74.00
Time per Move	2.20	2.86	2.37	2.34
Number of Moves	21.23	16.74	31.31	20.83
Toggles	6.11	3.96	9.70	3.93
Adjusted Undos	9.72	5.97	20.73	8.15

4.4.1 General Performance

Overall, participants found optimal solutions frequently with better performance in the Search condition of both problems. However, more search was required in general in the Search condition indicating that this difference in performance comes with a cost in terms of the amount of effort needed to find a solution. Table 4.4 is a summary of all performance measures and a summary of evidence found for these measures for all factors considered is found in Table 4.5. A detailed breakdown of these results follows.

Table 4.5: Summary of evidence found for performance measures . *** indicating very strong evidence ($BF > 150$), ** indicating strong evidence ($BF > 20$), * indicating positive evidence($BF > 3$), . indicating weak or no evidence ($BF < 3$). Extremely high Bayes factors ($BF > 1000$) are indicated by an additional !. The the model in the denominator is denoted by /. % Opt is the % Optimal solutions found. PAO is Percent above Optimal. %MaxMin is the percent maximal (Independent Set condition) or minimal (Vertex Cover condition) solutions found. Time/Move is the mean time per selection. Adj. Moves is the mean adjusted number of Moves performed to find a solution. Tog is the number of Toggles. Adj. Undos is the mean adjusted number of Undos performed.

Factors(s)	**% Opt**	**PAO**	**%Max Min**	**Time /Move**	**Adj. Moves**	**Tog**	**Adj. Undos**
Problem	*** !	*	*** /	*	*** /	* /	*** !
Goal	*** !	NA	*** !	.	*** !/	*** !	*** !
Problem × Goal	*	NA	*	.	***	** /	* /
Graph Type	*** !	*	*** !	**	*** !	*** /	**
Problem × Graph	** /	*	.	**	*** !	**	**
Goal × Graph	.	NA	*	**	*** !	*	**
Problem × Goal × Graph	**	NA	*	**	*** !	*	**
Group	**	.	**	* /	*** !	*	*
Graph × Group	**	**	**	**	*** !	***	***
Problem × Group	*	*	**	**	*** !	.	.
Goal × Group	.	NA	.	*	*** !	*	**
Problem × Goal × Group	*	NA	*	*** /	*** !	*	*
Problem × Graph × Group	**	*	*	**	*** !	**	**
Goal × Graph × Group	**	NA	**	**	*** !	**	**
All factors	*	NA	*	**	*** !	*	*

Hypothesis 1: Impact of Not-Euclidean Input on Performance

In general, performance on instances of the Optimization version of these two problems is good ($\mu = 10.16$) with slightly better performance in the Vertex Cover condition ($\mu = 9.60$) than in the Independent Set condition ($\mu = 10.74$). No Bayes factor analysis for these results is available because no other model exists within this experiment with which to compare, but these results will be compared to that on the Euclidean problem E-TSP in Section 5.1 below.

4.4.2 Maximizing vs. Minimizing

Hypothesis 2: Relative Performance on Quasi-Equivalent Maximization versus Minimization Problems

To investigate how performance differs between a maximization and a minimization problem performance is compared between the Vertex Cover and Independent Set problems. Bayes factor found very strong evidence for the Problem factor on most performance measures (% Optimal, % Maximal/Minimal, Number of Moves, and Adjusted Undos) as seen in Table 4.6. In terms of solution quality performance was better in the Independent Set condition with both more optimal solutions, and more maximal or minimal solutions found than in the Vertex Cover condition. This trend is still present when the Search and Optimization conditions are considered separately although there is notably less evidence for an interaction between Problem Version and Problem factors on the % Optimal solutions found.

Another measure of performance is the amount of search needed to find a solution as measured by the number of Moves and the number of Undos needed to find a solution. Less search was needed by participants in the Independent Set condition, as seen in Table 4.6, both in terms of the number of Moves and the number of Undos. This trend continues when each Problem Version is considered separately. Very strong evidence is found for the Problem factor on both these measures and for an interaction between Problem Version and Problem on the number of Moves. Strong evidence is found for this interaction on the number of Undos.

4.4.3 The Role of the Goal

Hypothesis 3: Impact of Infeasible-to-Evaluate Goal on Performance

To investigate the how an infeasible-to-evaluate goal impacts performance on these problems, results are compared between the Optimization version, with an infeasible-to-evaluate goal, and Search version, with a feasible-to-evaluate goal. This comparison is done for both problems together (Problem Version factor) as well as with each problem considered separately (the interaction between Problem Version and Problem). As seen in Table 4.7, when both problems were considered together or separately performance as measured by % Optimal and % Maximal/Minimal was better in the Search condition than in the Optimization condition. However, more search was required in the Search condition than in the Optimization condition, as measured by

Table 4.6: Performance comparison of maximization and minimization problems. Bayes factor analysis evidence for this model also presented. The column labeled PF indicates the evidence for the Problem Factor, and the column labeled PPF indicates the evidence for the interaction between Problem and Problem Version factors, with *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence.

	Both Versions			**Search**		**Optimization**		
Measure	**Ind. Set**	**Vertex Cover**	**PF**	**Ind. Set**	**Vertex Cover**	**Ind. Set**	**Vertex Cover**	**PPF**
% Optimal	70.09	55.30	***	83.75	70.13	57.23	40.67	*
% Maximal/ Minimal	93.03	84.62	***	95.00	96.0	91.0	74.00	***
Number of Moves	18.99	26.00	***	21.23	31.31	16.74	20.83	***
Adjusted Undos	7.847	14.36	***	9.72	20.73	5.97	8.15	**

both the number of Moves, and the number of Undos. Bayes factor analysis found very strong evidence for the Problem Version factor on both performance measures (% Optimal and % Maximal/Minimal) with better performance in the Search version when the two problems are considered together. Positive evidence was found for the interaction between Problem and Problem Version on the % Optimal solutions found and very strong evidence was found for this interaction on the % Maximal/Minimal solutions found, as seen in Table 4.7. Similarly, very strong evidence was found for the Problem Version factor and the interaction between Problem and Problem Version factors on the number of moves, and positive evidence was found for the interaction between Problem and Problem Version on the number of Undos.

Hypothesis 4: Goal Modification of Problems with Infeasible to Encode Goal and its Impact on Performance

Hypothesis 4 addresses not only how an infeasible-to-evaluate goal impacts performance but, more specifically, how goal modification impacts performance. If an infeasible to encode goal results in goal modification then it is expected that solutions to instances of problems where the goal is modified will better match solutions to a related problem with a feasible to identify goal than to the given task. A number of goal modifications were proposed for the problems used in this study. To evaluate

Table 4.7: Impact of infeasible-to-evaluate goal on performance. Bayes factor analysis evidence for this model also presented. The column labeled PVF indicates the evidence for the Problem Version Factor, and the column labeled PPF indicates the evidence for the interaction between Problem and Problem Version factors, with *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence.

	Both Problems			**Independent Set**		**Vertex Cover**		
Measure	**Search**	**Opt**	**PVF**	**Search**	**Opt**	**Search**	**Opt**	**PPF**
% Optimal	76.89	48.95	***	83.75	57.23	70.13	40.67	*
% Maximal/ Minimal	95.40	82.29	***	95.00	91.00	96.00	74.00	***
Number of Moves	26.23	18.76	***	21.23	16.74	31.31	20.83	***
Adjusted Undos	15.17	7.041	***	9.72	5.97	20.73	8.15	*
Path Length	1.769	1.919	**	1.99	2.05	1.55	1.79	.
Euclidean Distance	174.5	184.3	*	176.17	177.63	172.71	191.05	*

this hypothesis performance is compared to solutions on problems which might be the result of goal modification.

The proposed modifications were predicted to impact performance in terms of % Optimal solutions, % Maximal/Minimal solutions, and the amount of search needed to find a solution. The locality of selections is also evaluated as an indication of the use of local strategies. Performance in the Optimization condition of each problem separately is of main interest because it is unlikely that goal modification would be identical on the two different problems. As shown in Table 4.7, participants in the Optimization version of the Independent Set condition found optimal solutions more than half the time, found maximal solutions with high frequency, and did little search. In contrast, in the Optimization version of the Vertex Cover problem participants in the Optimization condition found optimal solutions less than half of the time, found a moderate number of minimal solutions, and performed some search. This hypothesis considers the same factors as Hypothesis 3 and therefore the evidence reported above applies here. Subsequent selections were closer as measured both by path and Euclidean distance in the Search Condition of both problems and in each problem separately. There is strong evidence for the Problem Version factor on Path

Distance and positive evidence for this factor on Euclidean Distance. There is weak evidence for an interaction between the Problem Version and Problem factors on Path Distance and positive evidence for this interaction on Euclidean Distance, as shown in Table 4.7.

Hypothesis 5: Impact of Goal Modification on Cognitive Load

Hypothesis 5 addresses how goal modification impacts the amount of cognitive load experienced by problem solvers. Cognitive load is measured by the amount of time needed to make individual move selections and the amount of Toggles performed. In terms of the amount of time needed on average to make move selections, slightly more time was taken per move in the Optimization condition. In the Independent Set condition more time is taken per move in the Optimization condition, and in the Vertex Cover condition there is nearly no difference in the time taken per move between the Problem Versions. Positive evidence is found for the Problem Version factor alone and weak evidence is found of an interaction between Problem Version and Problem on this measure. Details can be seen in Table 4.8. Participants in the Search condition used a higher number of Toggles with very strong evidence for the Problem Version factor on this measure and strong evidence for an interaction between Problem Version and Problem factors. Details can be found in Table 4.8.

Table 4.8: Impact of infeasible-to-evaluate goal on search and cognitive load. Bayes factor analysis evidence for this model also presented. The column labeled PF indicates the evidence for the Problem Factor, and the column labeled PPF indicates the evidence for the interaction between Problem and Problem Version factors, with *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence.

	Both Problems			**Independent Set**		**Vertex Cover**		
Measure	**Search**	**Opt**	PVF	**Search**	**Opt**	**Search**	**Opt**	PPF
Time per Move	2.284	2.603	*	2.20	2.86	2.37	2.34	.
Toggles	7.899	3.94	***	6.11	3.96	9.70	3.93	**

Hypothesis 6: Impact of Instance Type

To investigate the impact instance type, performance in terms of % Optimal and % Maximal/Minimal is compared on the different graph types, and any interactions

between Graph Type, Problem, and Problem Type are considered. As seen in Table 4.9, very strong evidence for a difference in performance is only found when Graph Type is considered alone, and then only on the % Optimal solutions found. There is less evidence for an interaction between Graph Type and the other factors of interest here. In terms of the % Optimal solutions found, performance differed between graph types, with highest mean % Optimal solutions found on D1 graphs, and lowest on Greedy Resistant and Random graphs, as shown in Figure 4.4 and Table 4.10. Very strong evidence was found for Graph Type on the percent of optimal solutions found. Positive evidence is found for an interaction between Graph Type and Problem and for an interaction between Graph Type, Problem and Problem Version, on both the % Optimal solutions. See Tables 4.10 and 4.9 for details. Performance differed little between Graph Type in terms of the % Maximal/Minimal solutions found, as seen in Table 4.10, with positive evidence found for an interaction between Graph Type and Problem Version on the % Maximal/Minimal solutions. There was more variation between Graph Type in the Optimization condition on this measure than in the Search condition. See Table A.4 for details. Positive evidence was also found for an interaction between Graph Type, Problem and Problem Version on the % Maximal/Minimal solutions found, as seen in Figure 4.5

Table 4.9: Summary of Evidence for the Impact of Instance Type. With *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence. Factor names are shortened for brevity. Graph refers to the Graph Type factor, Goal refers to the Problem Version factor.

Measure	**Graph**	**Graph × Problem**	**Graph × Goal**	**Graph × Problem × Goal**
% Optimal	***	*	.	*
% Maximal/ Minimal	.	.	*	*
Time/ Per Move	**	**	**	**
Moves	***	***	***	***
Undos	**	**	**	**
Toggles	***	**	*	*

Graph Type and all interactions with Graph Type strongly impact cognitive load and search, as seen in Table 4.9. Graph Type appears to impact the amount of time needed per move with the greatest Time per Move spent on NS graphs, and

Table 4.10: Impact of Graph Type on performance.

Measure	D1	D2	GR	NS	Rnd
% Optimal	79.74	64.8	55.75	59.26	54.32
% Maximal/ Minimal	93.95	88.28	86.14	89.21	86.55
Time Per Move	2.330	2.476	2.284	2.674	2.447
Moves	19.46	24.23	23.8	22.11	22.8
Undos	8.221	12.34	11.72	11.77	11.37
Toggles	3.296	6.837	6.288	6.159	7.007

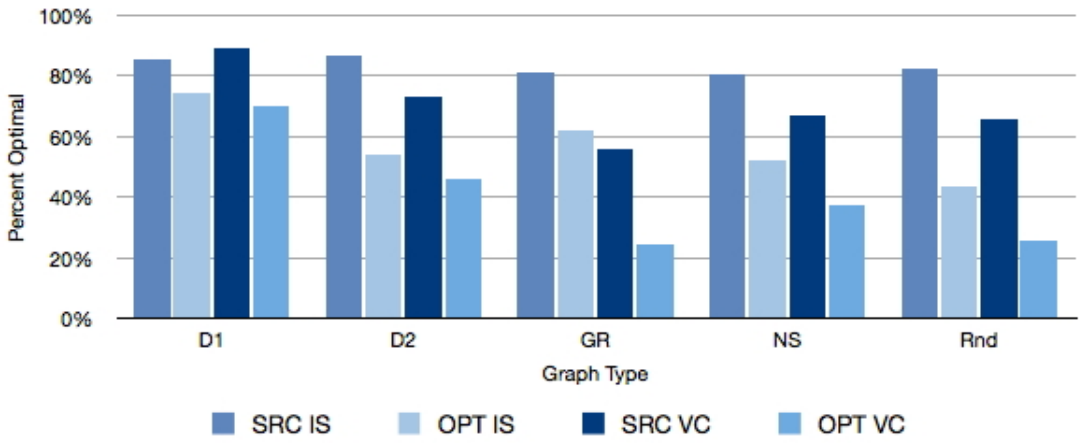


Figure 4.4: Percent Optimal solutions by Graph Type, Problem and Problem Version

the least spent on GR graphs. Strong evidence is found on this measure for Graph Type, for an interaction between Graph Type and Problem factor, for an interaction between Graph Type and Problem Version factor, and for the interaction between Graph Type, Problem and Problem Version. Details are found in Tables 4.9 and 4.10, and Figure 4.6. Graph Type impacts the number of moves and undos needed to find a solution, with the fewest Moves and Undos on D1 graphs, and the greatest number of Moves and Undos on D2 graphs. Strong and very strong evidence is found for these factors and their interactions on both these measures. Details can be seen in Table 4.9, and 4.10. The interaction of Graph Type, Problem, and Goal impacted the number of Moves as can be seen in Figure 4.7, and Undos in Figure 4.8. Graph Type impacted the number of Toggles, with the fewest Toggles on D1 graphs, and the most on Rnd graphs, as seen in Tables 4.10 and 4.9. The same general pattern is

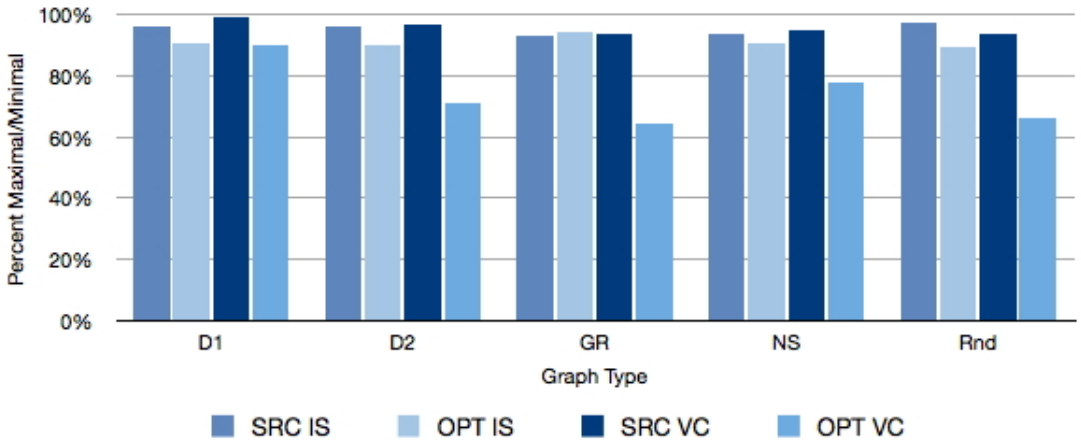


Figure 4.5: Percent Maximal/Minimal solutions by Graph Type, Problem and Problem Version

seen when the Independent Set and Vertex Cover problems are considered separately; however, more Toggles are used, in general, in the Vertex Cover condition, as shown in A.6. Very strong evidence is found for Graph Type, strong evidence is found on the interaction between Graph Type and Problem, and positive evidence is found for the remaining interactions, on the number of Toggles.

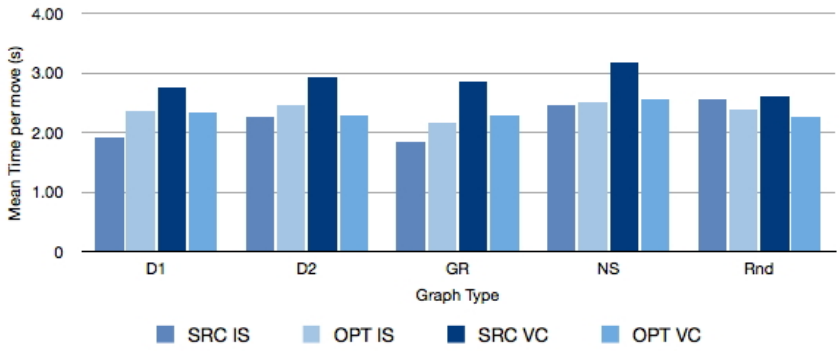


Figure 4.6: Mean Time per Move by Problem, Problem Version and Graph Type

Hypothesis 7: Impact of Instance Type Ordering

It has been observed in other studies that instance order can impact performance on problems of transformation, and to investigate whether this holds for the hard

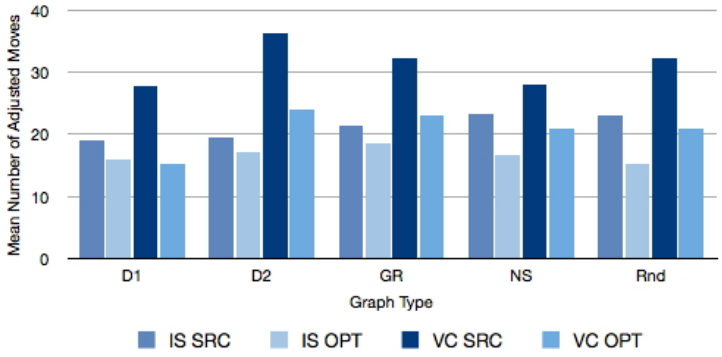


Figure 4.7: Mean number of adjusted Moves by Graph Type, Problem and Problem Version

computational problems used in this study, performance is compared between the 4 different instance orderings. Participants were assigned to one of 4 groups (A, B, C, or D), and in each group the D1, D2, NS and GR instances were ordered differently. Rnd graphs were always presented last. To evaluate how instance order impacts performance, the Group factor is considered in terms of how performance differs between groups.

Results show that Group (graph order) impacts performance, with strong evidence found for the % Optimal solutions found and positive evidence for the % Maximal/Minimal solutions, as seen in Table 4.11. Group D found optimal solutions less often, and Group A found them most frequently. While the same pattern does not follow for the % Maximal/Minimal solutions found, overall Group A still found the most Maximal/Minimal solutions of all groups. Group, or graph order, also impacts the amount of search and cognitive load, with the most pronounced impact on the number of Moves needed to find a solution. Group A used the most moves, and Group B used the fewest, with very strong evidence for this factor on this measure, as seen in Table 4.11. Less evidence is found for the other measures of cognitive load and search, and no consistent pattern is observed for these measures.

4.4.4 Strategy Acquisition

While this research study was not originally designed to investigate strategy acquisition, the instances that were generated to either be vulnerable or resistant to specific strategies provide an opportunity to attempt to better understand how and if these

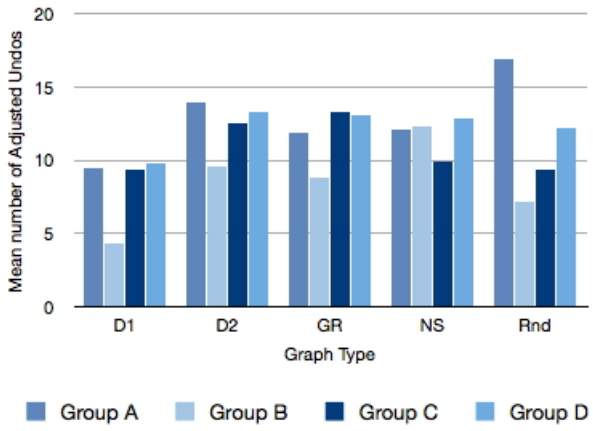


Figure 4.8: Mean number of adjusted Undos by Problem Version, Problem and Graph Type

strategies were acquired and applied, and any possible impact of this strategy acquisition. A summary of evidence for strategy measures is presented in Table 4.12, and a summary of all strategy measures is presented in Table 4.13. In the following subsections, results are presented that support the specific hypotheses proposed above that address strategy acquisition.

Table 4.11: Impact of graph order on performance. The column labeled Group shows the evidence found for each measure, with *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence.

Measure	**A**	**B**	**C**	**D**	Group
% Optimal	79.74	61.77	62.75	60.71	**
% Maximal/ Minimal	93.95	86.79	87.33	89.62	*
Time Per Move	2.299	3.028	2.207	2.239	*
Moves	24.28	19.74	22.16	22.8	***
Undos	12.85	8.411	10.87	12.22	*
Toggles	5.612	4.982	6.296	6.778	*

Table 4.12: Summary of evidence for strategy measures. *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence. Path distance is the mean length of the shortest path between selections. E-distance is the mean Euclidean Distance between selections. Degree is the mean Degree of selections. D1 at Onset is the mean normalized number of D1 at Onset selections. D1 is the mean normalized number of D1 selections. D2 at Onset is the mean normalized number of D2 at Onset selections. D2 is the mean normalized number of D2 selections.

Factors(s)	**Path Dist.**	**E-dist.**	**Degree**	**D1 at Onset**	**D1**	**D2 at Onset**	**D2**
Problem	*** !	.	*** !	*** /	*** !	*** !	*** !
Goal	** /	* /	*** !	.	.	*	.
Problem × Goal	.	.	** /	.	.	.	*
Graph Type	*** !	*** !	*** !	*** !	*** !	***!	*** !
Problem × Graph	*** !/	**	*** !	* /	*** !	*** !	*** !
Goal × Graph	*	**	*	.	**	.	*
Problem × Goal × Graph	*	*	*	.	*	.	*
Group	*	* /	* /	*	.	*	**
Graph × Group	**	**	***	*	**	.	*
Problem × Group	*** /	*** !	*** !	.	*/	.	.
Goal × Group	**	*	***	.	**	*	*
Problem × Goal × Group	*	*	*	*	*	*	*
Problem × Graph × Group	**	**	**	*	*	*	.
Goal × Graph × Group	**	**	**	.	**	*	.
All factors	**	*	*	.	*	.	.

Table 4.13: Summary of strategy results for all conditions. Path distance is the mean length of the shortest path between selections. E-distance is the mean Euclidean Distance between selections. Degree is the mean Degree of selections. D1 at Onset is the mean number of D1 at Onset selections. D1 is the mean number of D1 selections. D2 at Onset is the mean number of D2 at Onset selections. D2 is the mean number of D2 selections.

	Independent Set		**Vertex Cover**	
	Search	**Optimization**	**Search**	**Optimization**
Path Length	1.99	2.05	1.55	1.79
Euclidean Distance	176.17	177.63	172.71	191.05
Degree	1.96	2.01	2.52	2.82
D1 (Onset)	5.48	4.67	6.40	4.17
D1	2.53	2.05	14.28	8.44
D2 (Onset)	3.43	2.75	7.81	4.60
D2	3.19	2.70	2.47	1.60

Hypothesis 8: Ability to Acquire Correct Strategies

If problem solvers are able to acquire and apply the correct D1 and D2 strategies, there should be a large proportion of selections that could be D1 or D2 selections. There were two kinds of D1 and D2 vertices, those that were available at onset, and those that were the result of the modified state of the graph at the time of the selection. The Problem Version factor is of main interest here, as differences in participants' ability to acquire and apply these correct strategies are predicted to be impacted by participants' ability to determine if a strategy was successful. The feedback of the given goal in the Search condition could give indication of the correctness of the selection. The interaction between Problem Version and Problem is presented to evaluate how strategy acquisition and application might differ between the two problems and versions.

When mean counts are considered, there is a marked difference in the number of both D1 and D2 selections in the Search and Optimization conditions. However, since participants in the Search condition make more selections, the counts of these D1 and D2 selections were normalized by the total number of selections, and the mean normalized D1 and D2 selections show nearly no pattern of difference between either the Search and the Optimization condition, nor between Problems, as shown in Table 4.14. Further supporting this lack of difference is a lack of evidence based on Bayes factor analysis. No evidence is found for the Problem Version factor for D1 or D2 selections, with the exception of positive evidence for normalized D2 at onset selections. Similarly, no evidence is found for an interaction between Problem and Problem Version factors, except on normalized D2 selections, as seen in Table 4.14.

Table 4.14: D1 and D2 vertex selections. Counts are presented only for instances which have the associated vertex type at onset. For D1 selections instances D1 and Rnd are considered. For D2 selections instances D2 and Rnd are considered. All values are normalized mean counts. Bayes factor analysis evidence for this model also presented. The column labeled PF indicates the evidence for the Problem Factor, and the column labeled PPF indicates the evidence for the interaction between Problem and Problem Version factors, with *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence.

	Both Versions			**Independent Set**		**Vertex Cover**		
Measure	**SRC**	**OPT.**	**PVF**	**SRC**	**OPT**	**SRC**	**OPT**	**PPF**
D1 (Onset)	5.918	4.416	***	5.45	4.66	6.40	4.17	*
D1	8.369	5.236	***	2.49	2.05	14.28	8.44	***
D1 (Onset) Normalized	0.2425	0.2707	.	0.26	0.30	0.22	0.24	.
D1 Normalized	0.2518	0.2734	.	0.11	0.12	0.39	0.42	.
D2 (Onset)	5.604	3.664	***	3.42	2.75	7.81	4.60	*
D2	2.828	2.151	***	3.19	2.70	2.47	1.60	.
D2 (Onset) Normalized	0.1803	0.1795	*	0.1752	0.1843	0.1836	0.1764	.
D2 Normalized	0.09315	0.08857	.	0.15	0.14	0.03	0.03	*

Hypotheses 9 and 11: Impact of Acquiring Correct or Heuristic Strategies

To evaluate the impact of acquiring correct or heuristic strategies, performance is compared between instance type for different block order (or Group), and the interaction between Group and Graph Type, in particular, is of interest. To evaluate differences between this impact based on Problem and Problem Version, the interactions between all four of these factors (Graph, Group, Problem, and Problem Version) are presented. When the interaction between Graph Type and Group are considered, there is strong or very strong evidence found for all performance measures as well as for all measures of search or cognitive load, as seen in Table 4.5. Performance as measured by % Optimal solutions on Rnd instances differs little between groups. However performance varies between groups on D1 and D2 instances, with best performance in Groups C and D on D1 instances, and best performance in Groups A and C on D2 instances.

As seen in Table 4.15, there is evidence that instance and order impact performance, and additionally, that these differences manifest themselves differently across

Table 4.15: Summary of evidence for acquiring correct or heuristic strategies. *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence. Bayes factors over 1000 are indicated by an additional !.

Factors	**% Opt**	**%Max Min**	**Time /Move**	**Adj. Moves**	**Tog**	**Adj. Undos**
Graph × Group	**	**	**	***!	***	***
Graph × Group × Problem	**	*	**	***!	**	**
Graph × Group × Goal	**	**	**	***!	**	**
All Factors	*	*	**	***!	*	*

the two problems and their versions. Strong evidence is found, on % Optimal solutions, for an interaction between Graph Type and Group, for an interaction between Graph Type, Group, and Problem, and for an interaction between Graph Type, Group, and Problem Version. Overall, performance on D1 instances was best, with better performance in the Search Condition. When the final block of instances is considered (Rnd instances), the pattern seen in Figure 4.9 shows that in both the Search and Optimization conditions, participants in Group A performed best. However, the performance of the remaining three groups on these same instances shows a reverse effect between the Search and Optimization conditions. Slightly less evidence is found for these interactions on the % Maximal/Minimal solutions found, with strong evidence only on the interaction between Graph Type and Group, and the interaction between Graph Type, Group, and Goal. Performance on this measure follows a similar trend, as seen in Figure 4.10, as that observed for the % Optimal solutions.

There was a great deal of evidence found for this interaction on measures of search. Most notably, very strong evidence was found for all four interactions for the number of adjusted moves. See Figures 4.11, 4.12, 4.13, and 4.14 for a visual representation of how these factors impacted the amount of search needed. Very strong evidence was found, on the mean number of adjusted undos, for the interaction between Graph Type and Group. Strong evidence was found on this measure for the interaction between Graph Type, Group, and Problem and for the interaction between Graph Type, Group, and Problem Version. Positive evidence was found for an interaction between all factors on this measure. Details of all results can be found in Table A.8, and the overall pattern is not unlike that seen for the number of moves.

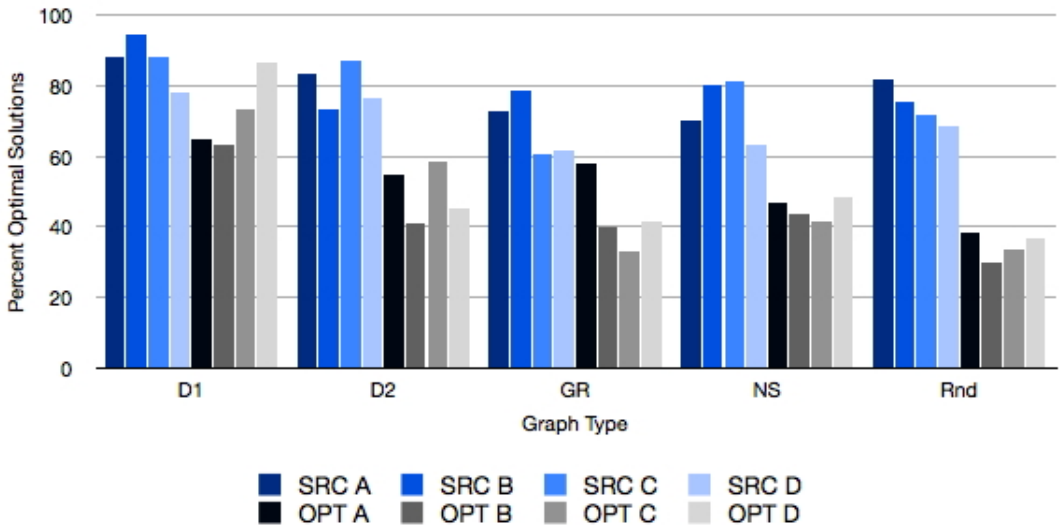


Figure 4.9: Percent Optimal solutions found by Problem Version, Graph Type and Group

Finally, there is strong evidence that instance order and instance type impact the amount of cognitive load, measured both in terms of the number of Toggles, as well as in the amount of time needed to make each selection. Very strong evidence was found for an interaction between Graph Type and Group on the mean number of Toggles, strong evidence was found for the interaction between Graph Type and Group, and each of Problem and Problem Version, separately. Positive evidence was found for the interaction of all factors on this measure. Details of how these factors impacted this measure can be seen in Figures 4.15, 4.16, and 4.17. As seen in Table 4.15, strong evidence was found for all four interactions on the Time per Move. Details of how these factors impacted the time taken per move can be seen in Figures 4.18, 4.19, and 4.20.

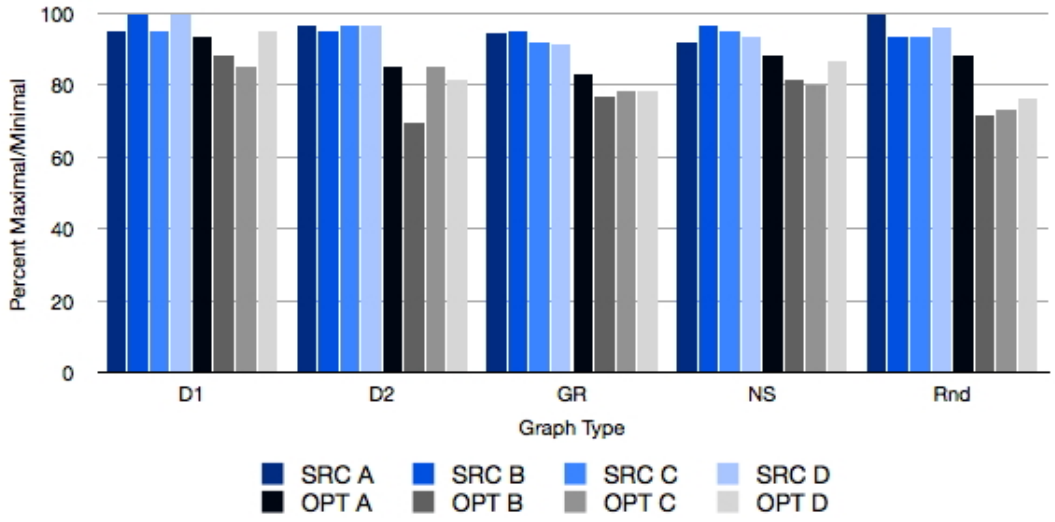


Figure 4.10: Percent Maximal/Minimal solutions found by Problem Version, Group and Graph Type

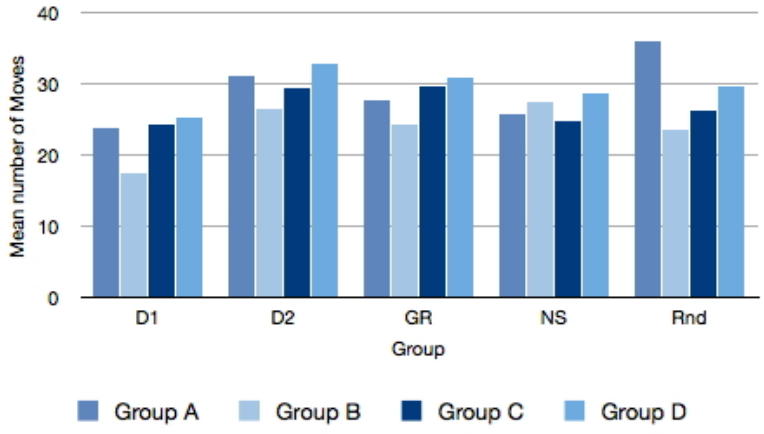


Figure 4.11: Mean number of adjusted Moves by Group and Graph Type

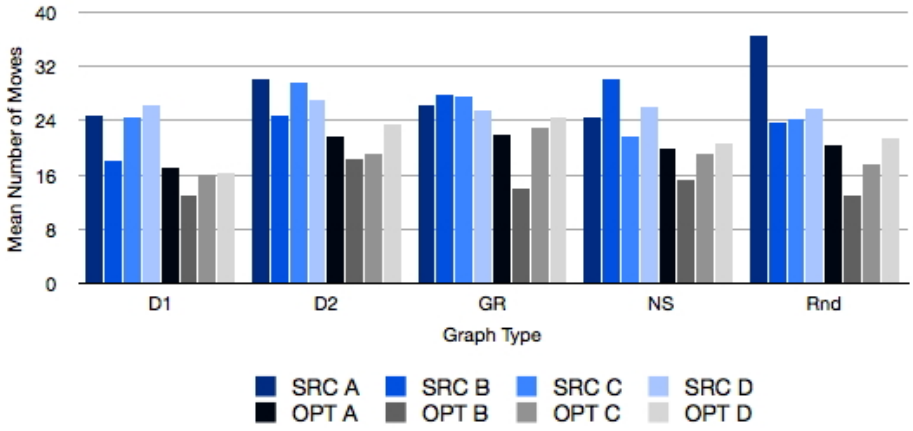


Figure 4.12: Mean number of adjusted Moves by Problem Version, Group and Graph Type

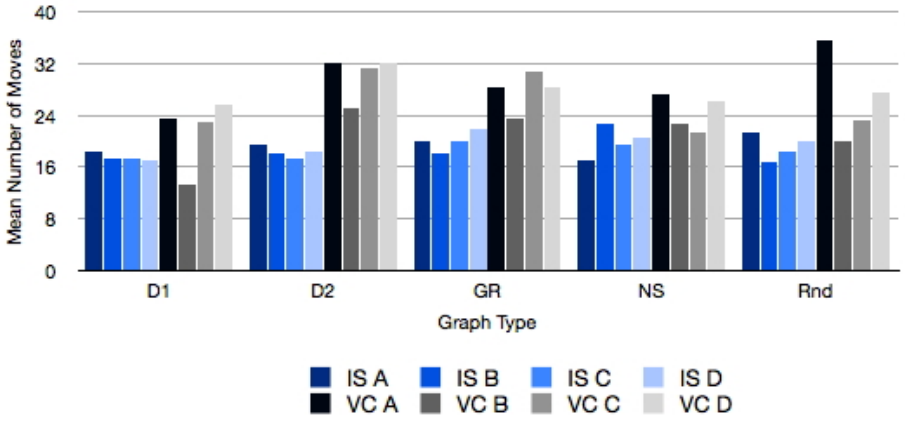


Figure 4.13: Mean number of adjusted Moves by Problem, Group and Graph Type

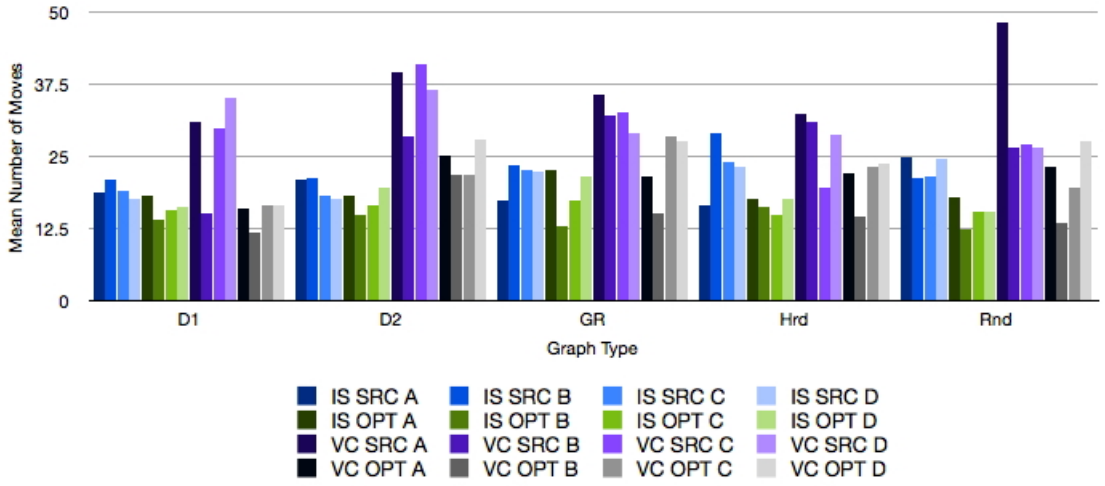


Figure 4.14: Mean number of adjusted Moves by All Factors

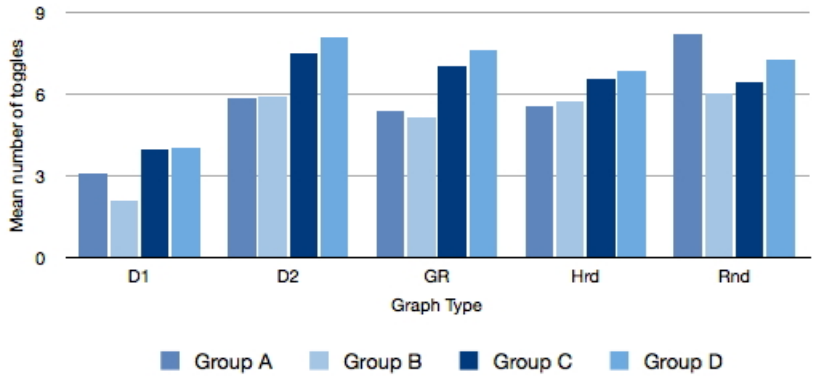


Figure 4.15: Mean number of Toggles by Graph Type and Group

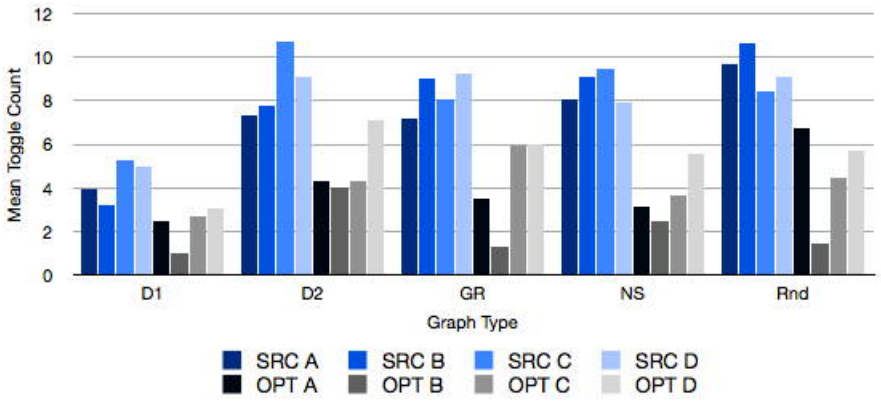


Figure 4.16: Mean number of Toggles by Problem Version, Graph Type and Group

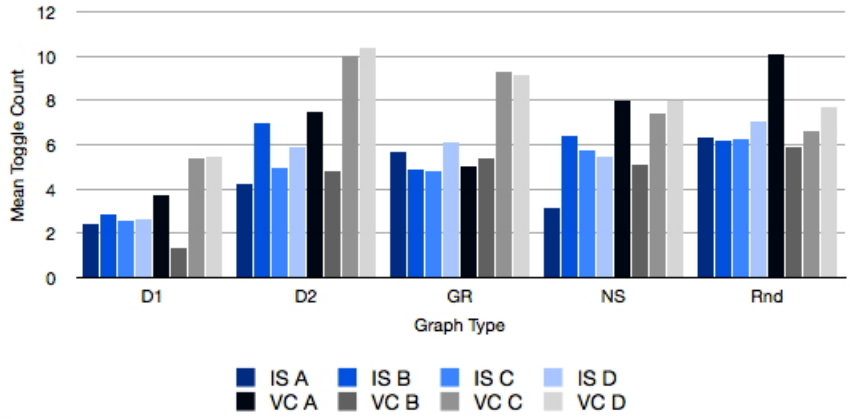


Figure 4.17: Mean number of Toggles by Problem, Graph Type and Group

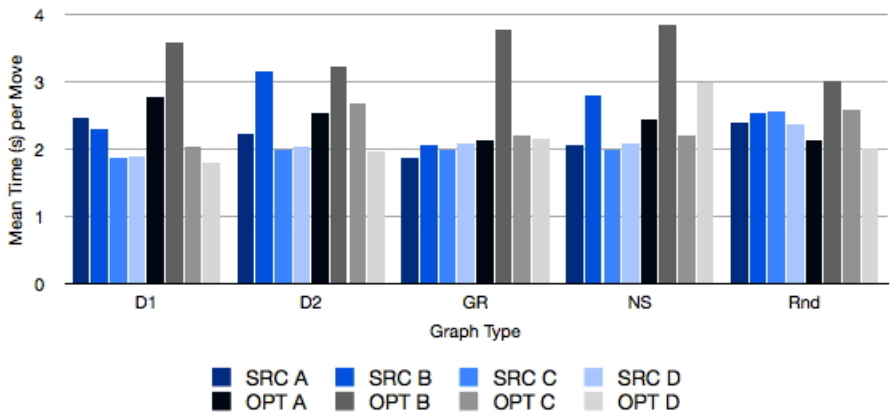


Figure 4.18: Mean Time per Move by Problem Version, Graph Type and Group

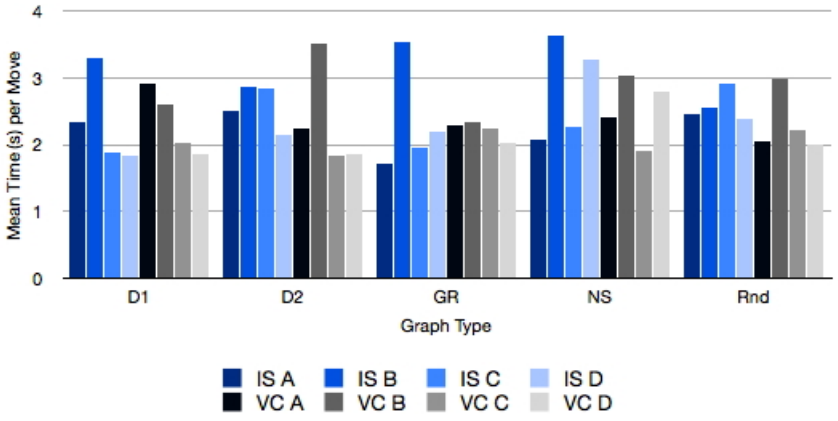


Figure 4.19: Mean Time per Move by Problem, Graph Type and Group

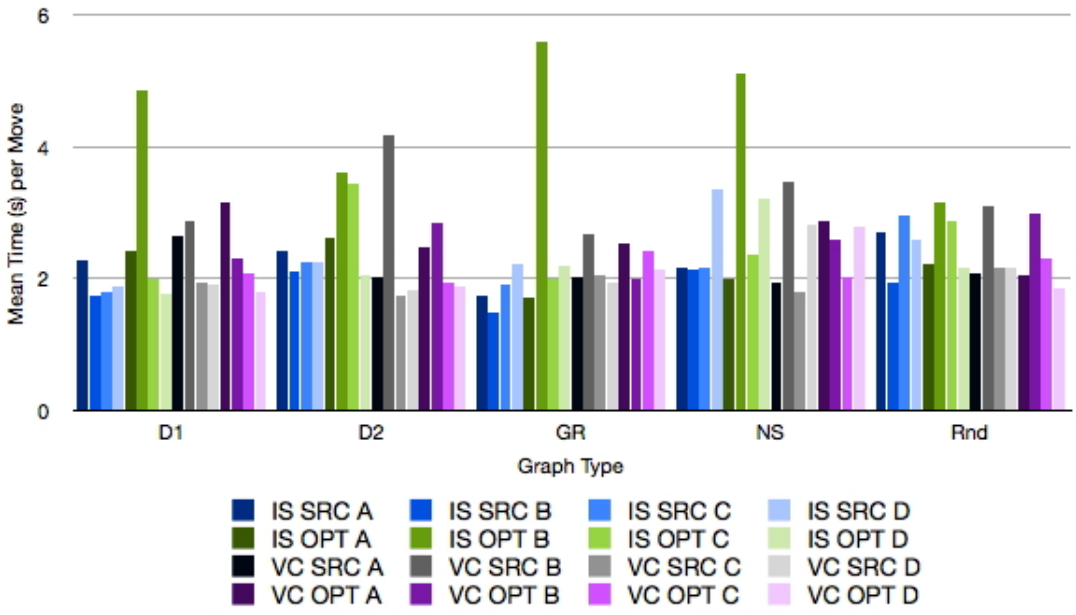


Figure 4.20: Mean Time per Move by all factors

Hypothesis 10: Ability to Avoid the Application of Heuristic Strategies

If problem solvers are able to avoid the heuristic GR strategies, there should be lower degree selections in the Vertex Cover condition, and higher degree selections in the Independent Set condition, than if GR strategies are unsuccessfully avoided. Like with the correct strategy, differences in participants' ability to avoid this heuristic strategy is predicted to be impacted by participants' ability to determine if a strategy was unsuccessful, and therefore the Problem Version factor is of interest. Mean degree of selections is higher in the Optimization condition than in the Search condition, when both problems are considered together, as well as when they are considered separately, with very strong evidence for the Problem Version factor, as seen in Table 4.16. Mean degree is higher overall in the Vertex Cover condition than in the Independent Set condition, with very strong evidence for an interaction between Problem and Problem Version, as also shown in Table 4.16.

Table 4.16: Degree of vertex selections. Mean degree is presented. Bayes factor analysis evidence for this model also presented. The column labeled PF indicates the evidence for the Problem Factor, and the column labeled PPF indicates the evidence for the interaction between Problem and Problem Version factors, with *** indicating very strong evidence, ** indicating strong evidence, * indicating positive evidence, . indicating weak or no evidence.

	Both Versions			**Independent Set**		**Vertex Cover**		
Measure	**SRC**	**OPT.**	**PVF**	**SRC**	**OPT**	**SRC**	**OPT**	**PPF**
Degree	2.244	2.418	***	1.97	2.01	2.52	2.82	***

4.5 Summary

In this Chapter, the main results of the research study were presented. It was found that in general, performance on the Optimization versions of these two hard not-Euclidean computational problems was in keeping with previously reported performance results on other hard optimization version, with participants finding solutions that were roughly within 10% of optimal. This experiment considered a number of other factors, including Problem, Problem Version, Graph Type, and Group (graph order). Strong evidence was found for these factors and their interactions on most measures.

Evidence was found for differences between the two problems used, Vertex Cover and Independent Set, on all measures: performance, search, and strategies. Performance was better in the Independent Set condition than on the Vertex Cover condition, on all performance measures except PAO, where Vertex Cover performance was slightly better. Interestingly, the amount of search used in the Independent Set condition was less than in the Vertex Cover condition, both in terms of the number of Moves, and the number of Undos. Finally, strategy selections appear to differ between these two problems, on all measures except Euclidean Distance.

Performance differences emerged between the Search and Optimization problems, with performance as measured by the percent of optimal solutions better in the Search condition. In addition, the amount of search required to find solutions differed between Problems, with more search in the Search version, corresponding with more optimal solutions. Cognitive load appears to be higher in the Search condition, as indicated by the greater number of Toggles. In contrast, little difference in move selection strategy emerges from this analysis, with evidence of differences only in terms of Path Distance and Degree.

Evidence was found for differences in performance between the different graph types, with D1 graphs appearing to be easiest, and GR and Random graphs more challenging. Graph Type appears to greatly impact the amount of search needed, as well as cognitive load. These results are consistent, in that better performance on D1 graphs is associated with less search, fewer Toggles, and less time per selection. Strategy application also appears to be influenced by Graph Type, with strong evidence on all measures of Move Selection strategies between different graph types.

Finally, a great deal of evidence was found for differences on all measures as a result of Graph order. These results are quite complex to interpret, due to the great number of factors and interactions that must be considered. Deeper analysis follows in the upcoming Chapter.

Overall, some very interesting results emerged from this research study, with strong evidence for a number of factors which have not previously been considered in studies of human performance on hard computational problems. These results are analyzed in detail in Chapter 5 in the context of the hypotheses presented in Chapter 3.

Chapter 5

Discussion

Strong evidence was found in Chapter 4 for differences in performance on all major measures between goal well-definedness giving preliminary evidence for the theory that the goals of the problem are encoded differently between these conditions. Similarly, the results support the theory that there may be performance differences, either between minimization and maximization problems, or between the Vertex Cover and Independent Set problems. A number of hypotheses were presented at the end of Chapter 3. The results presented in Chapter 4 are analyzed in greater detail in light of these hypotheses.

General performance results are presented first. Then performance differences between conditions are analyzed against the theory that the well-definedness of the goal impacts how the goal is encoded, and therefore the problem that is being solved. A finer analysis will then attempt to find evidence to support the proposed goal modifications presented in Chapter 3. If goal well-definedness results in problem modification, then it is possible that the problems being solved in the different conditions will result in different levels of cognitive load. This possibility is examined by looking at differences in the use of cognitive aids to support problem solving. Performance differences between the Vertex Cover and Independent Set problems is compared to see if either maximization or minimization problems are subjectively easier, or if they are equally challenging.

Hard computational problems are unlikely to be solvable by general problem-solving strategies. However, some instances of these hard problems can be solved, or reduced to a small kernel or reduced instance, through the application of simple strategies. Graphs were designed to be vulnerable or resistant to known strategies. Analysis of performance measures, and the properties of vertices selected, will at-

tempt to determine if participants are able to acquire and apply these strategies, and how instance order and selection impacts performance. Manipulation of the order that different graph types are presented may help determine what factors support or interfere with strategy acquisition and application.

5.1 General Performance

An original motivation of this work was to compare human performance on hard not-Euclidean optimization problems to that on other hard optimization problems like E-TSP. Results from the Optimization conditions of both the Vertex Cover and Independent Set conditions are in keeping with previously observed performance, with mean PAO close to 10% for both problems. Interestingly, however, performance on the Search versions of these same problems is better than the Optimization versions, measured as percent optimal solutions found. This raises the possibility that previously reported human performance results on E-TSP might be improved upon if participants are tasked with the Search version of that problem.

These general performance results suggest that visual processing alone is likely not responsible for the good quality previously noted on other hard optimization problems. The Gestalt principles, or the hierarchical pyramid model which have been proposed as being potentially partially responsible for solution quality, may not be applicable on the not-Euclidean problems used in this study, and therefore cannot explain performance on the Vertex Cover and Independent Set problems.

5.2 Impact of Goal Well-Definedness

If participants in the Optimization condition are able to encode the goal exactly as it is given, then there should be no significant difference in performance on the Search and Optimization conditions of the same problem. This, of course, assumes that participants in the Search condition are able to encode the goal and are working on the task given. This possibility is investigated by considering how often optimal solutions are found in the Search condition. Participants are given the exact same instances to solve, with identical start states and legal operators, and the instances differ only in terms of the specificity of the given goal. If participants in the Optimization condition are unable to encode the given goal, or encode it differently than in the Search condition, and encode some other goal, then it is expected that there be

a difference in performance across the two versions as a result of the difference in what constitutes a goal state between the two versions. Performance as measured by percent optimal solutions supports Hypothesis 3: overall, performance by this measure is better on the Search version than on the Optimization version, as seen in Section 4.4.3. Within each Problem condition, performance is also better on the Search version than on its associated Optimization version.

Another measure of performance is the amount of backtracking needed to find a solution. Differences in the amount of backtracking needed to find a vertex cover or independent set could be indication of different goal states being encoded and searched for. Other possible explanations exist too, of course. The mean number of undos is much higher in the Search condition than in the Optimization condition, both across both problems, and within each problem. As shown in Table 4.4, this difference is much more pronounced in the Vertex Cover group, where the mean number of undos in the Search condition is over twice that in the Optimization version. One explanation for this difference is that relative to their associated Search versions, the problem(s) being solved by participants in the Optimization condition of the Vertex Cover problem is much easier than that being solved in the Independent Set problem.

5.3 Evaluation of Candidate Goal Modification

With these findings in support of the hypothesis that participants in the Optimization condition are solving a problem other than the one given, the next task is to attempt to narrow down alternate problems which could explain performance. In Section 3.1.3 four different goal modifications were identified for the Minimum Vertex Cover and Maximum Independent Set problems, which might explain performance, found in Tables 3.2 and 3.3 respectively. For each candidate modification, predicted results are compared to:

1. the quality of solutions found in the Optimization version,
2. the percent of minimal (VC) or maximal (IS) solutions found,
3. the percent optimal solutions found,
4. how much backtracking takes place.

These results can help narrow down which proposed modifications might explain performance on the Optimization versions of the problems given.

5.3.1 Minimal/Maximal Modification

One candidate modification is that of finding a minimal solution to the Vertex Cover problem or a maximal solution to the Independent Set problem. Recall that this modification predicts sub-optimal solutions, many minimal/maximal solutions, few optimal solutions, and little backtracking. The PAO found in both problems supports this modification with the mean PAO near 10% in both problems. The percent optimal solutions found in the Optimization condition is also much lower than in the Search version, as predicted. And finally, very little backtracking is done in the Optimization condition which further supports this modification.

The percent minimal/maximal solutions can be compared to the percent of optimal solutions to test which goal, optimization or minimal/maximal, better fits the data. These comparisons cannot prove that this restructuring modification is taking place, however, it can eliminate a candidate, or indicate that it is still a possibility. The percent of minimal solutions found in the Optimization version of the Vertex Cover problem is significantly higher than the percent optimal solutions in the same condition, and therefore is a better fit. These results, found in Section 4.4.3 above, strongly support the hypothesis that the infeasible-to-encode goal of the Optimization version of this problem could be restructured to that of a minimal solution. Similarly, the percent of maximal solutions found in the Optimization version of the Independent Set problem is significantly higher than the percent of optimal solutions in the same problem. Alternatively, it cannot be excluded that the problem could have been modified to another problem whose results coincidentally are also maximal/minimal with high likelihood.

5.3.2 Any Valid Cover/Independent Set

The second modification described discarded all sense of optimality altogether, and amounts to finding any valid vertex cover or independent set. Recall that this modification predicts sub-optimal solutions, non-minimal/maximal solutions, few optimal solutions, and no backtracking. The PAO found in the two versions of both problems matches the prediction of sub-optimal solutions in the Optimization version. In the Vertex Cover condition, the relatively lower mean percent minimal solutions may be indication that this modification is taking place. As well, very few optimal solutions are found (fewer than 50%), which could be support for this modification. Finally, the number of undos deviates from the total lack of undos predicted for this mod-

ification. In the Independent Set condition, the high number of maximal solutions found in the Optimization condition does not match the predicted performance for this modification. The mean percent optimal solutions is slightly over 50%, but still well below the number found in the associated Search condition also does not match this modification. Finally, the number of undos deviates from the total lack of undos predicted for this modification. In summary, there is mixed support for this modification for the Vertex Cover condition, and very little support for it in the Independent Set condition.

5.3.3 Local Optimization

Goal modification could also take the form of local optimization. Recall that this modification predicts sub-optimal solutions, non-minimal/maximal solutions, few optimal solutions, and a moderate amount of backtracking. In the Vertex Cover condition, PAO matches the prediction of sub-optimal solutions in the Optimization version, and the low percent optimal vertex covers found in the Vertex Cover condition supports this modification. The number of minimal vertex covers, however, is lower in this condition than in the Independent Set condition, and is weaker support for this modification. In the Independent Set condition PAO matches the prediction of sub-optimal solutions in the Optimization version. In addition the high number of maximal independent sets found, and higher (over 50%) percent optimal independent sets found, supports this modification. Another possible measure that could support this modification is how local subsequent moves are. The average path distance between moves in all conditions is small, hovering around a length of 2, which is further support of local optimization as a candidate modification. However, interestingly, the closeness of subsequent moves is even more pronounced in the Search version than in the Optimization version, which suggests that either local choices drive Move selections in these kinds of graph problems, independent of local optimization, or that local optimization is useful in attempting to solve both versions of these problems. In summary, mixed support is found for local optimization in both the Vertex Cover and Independent Set conditions.

5.3.4 Generation of Specific k-values

The final modification proposed was the Guess and Check modification, which predicted near-optimal solutions, many minimal/maximal solutions, many optimal solu-

tions, and an amount of backtracking similar to that seen in the Search conditions. The PAO in both problem conditions does not match the near-optimal solutions predicted by this modification. Similarly, the relatively low percent optimal vertex covers found in the Vertex Cover condition and independent sets found in the Independent Set condition does not match the optimality predicted. Finally, the low amount of undos in the Optimization versions of both problems relative to their associated Search versions also does not support this modification. These results do not strongly support the likelihood that this modification takes place with significant frequency.

5.4 Impact of Goal Modification on Cognitive Load

It is predicted that, tasked with a problem with an aspect of the goal that is not encodable, that problem solvers modify the goal to render it encodable. It was assumed that this modification should take place as parsimoniously as possible, that is by modifying the problem as little as possible. It seems likely that this modification should also result in a problem that is not harder than an encodable version of the task (for example the Search version). However, it is important to note that this need not be the case. The Optimization version of the problem given could feasibly be converted into a problem that is at least as hard as the Search version of the respective problem, and still be encodable. Recall the proposed modification in which increasingly smaller (or in the case of the maximization problem, Independent Set, larger) k values are tested until the optimal solution is found. If this modification took place, however, the number of moves needed to find a solution would be expected to be similar in the Optimization version to those used in the Search condition (which it is not), and for the percent of optimal solutions to be closer in the Optimization version, to that found in the Search condition (which it is not).

One possible consequence of modifying the problem into an easier one which is that it would likely take fewer moves to find a solution, and indeed this is what was observed in this study. In the Optimization version of the Independent Set condition, the mean number of moves to find a solution was lower than that in the Search version. Similarly, in the Optimization version of the Vertex Cover condition, the mean number of moves was fewer than that used in the Search version. These results strongly support the hypothesis that participants in the Optimization condition are solving an easier problem. Participants in the Optimization condition consistently use fewer moves to find a solution, which could be indication that the goal, whatever

it might be, is easier to find than in the Search condition.

Toggles are likely an indication of using the interface to gain information about the current and near future states of the problem, then reduced cognitive load could manifest as fewer Toggles. Indeed, many fewer toggles were observed in the Optimization condition than the Search condition, overall, as well as for each problem considered separately (see Table 4.4). These results strongly support the hypothesis that participants in the Optimization condition are solving a problem that is easier than the Search version, and that places less cognitive load on the system.

5.5 Performance Differences on Maximization and Minimization Problems

One of the goals of this research was to address the lack of research comparing performance on hard minimization and maximization problems. It is interesting to wonder why research thus far has focused on minimization problems. At the surface there is support for the conjecture that minimization problems are easier for the cognitive system to cope with than maximization problems. Maximization problems, without external memory aids, add an additional space complexity in needing to maintain a growing candidate solution, whereas with minimization problems the candidate solutions can become smaller as the goal is approached. Given the limits of the cognitive system, minimization problems could be more natural. In addition to this consideration, that minimization might put less strain on working memory than maximization, the k-Vertex Cover problem used in this study might be easier for people to solve due to it being fixed parameter tractable. The Vertex Cover problem is known to be fixed parameter tractable, whereas Independent Set is not[1]. This would seem to imply that Vertex Cover is somehow easier, as long as reduction rules can be learned and applied.

These two observations support the hypothesis that the Vertex Cover problem should be subjectively easier than the Independent Set problem. The results of this study, however, do not support this Hypothesis 2. The mean percent optimal solutions is higher in the Independent Set condition than in the Vertex Cover condition,

[1]While it is true, because of its relationship to Vertex Cover that the Independent Set problem is fixed parameter tractable for the parameter $p = n - k$, this does not have the same time complexity consequences, because as n grows it also causes the exponential component of the time requirements to grow in terms of n.

and this trend carries on when the Search and Optimization versions are considered separately. The percent minimal/maximal solutions continues with the pattern, with better performance on this measure in the Independent Set condition than in the Vertex Cover condition. In contrast, the mean PAO is higher in the Optimization versions of the Independent Set condition than in the Vertex Cover condition. These are slightly conflicting results. Perhaps whatever modification occurs in the Optimization condition of Independent Set problem results in maximal solutions, but further from optimal. These results are weak evidence too, and may not be indication of a strong effect. Recall, however, that the Greedy Resistant graphs were not as resistant to the greedy strategy in Independent Set condition as they were in the Vertex Cover condition. If these instances are removed from the analysis, and consider the percent optimal solutions on the remaining graph types, the trend is less pronounced. While the percent optimal independent sets is still higher in the Independent Set condition than the percent optimal vertex covers in the Vertex Cover condition Bayes factor analysis showed no evidence for these results and so they do not support or refute the observed difference in performance.

In summary, it appears that there is a difference in performance between these two problems, which may be a result of their being minimization or maximization versions, or an artifact of some other feature of these two problems. These results are far from conclusive: however the extreme difference in performance and the strong support for this model seem to indicate that the Independent Set problem is easier than the Vertex Cover problem. It also suggests that perhaps maximization is not as difficult as minimization, which warrants further investigation. Perhaps most interesting, however, is that these results are seemingly contradict what is thought to be known about the relative difficulty of these two problems based on their fixed parameter tractability. It is important to note, however, that the two reduction rules used to generate graphs for two of the blocks were equally applicable to both problems. It may be that a different choice of graphs would result in different relative performance results.

5.6 Strategy Acquisition

Evaluating whether or not problem solvers were able to acquire and apply correct or heuristic strategies poses a number of challenges. A given Selection can typically be explained by more than one strategy. For this reason, graphs were purposely

designed to be vulnerable to the identified correct strategies (D1 and D2), to aid in the evaluation of whether the strategies were applied. A different approach is used to evaluate Greedy strategies, based on the assumption that this strategy is being applied, and therefore graphs were designed to test if the Greedy strategy could be avoided.

5.6.1 Correct Strategies

Acquisition and application of the D1 strategy should result in better performance on instances where they can be used (possibly in conjunction with a small amount of exhaustive search) to solve exactly. The D1 strategy could be used to solve all D1 graphs in conjunction with a small amount of exhaustive search, and all Random graphs in conjunction with the D2 strategy and a small amount of exhaustive search. Exhaustive search is considered to be feasible if it can be solved within the constraints of working memory; that is, only a few alternatives must be considered to find the best moves. For example, consider Graph 2 shown in Figure 5.1. In the Vertex Cover condition, after all D1 selections have been made, the remaining reduced instance (shown in Figure 5.2) is small. Exhaustive search is likely feasible on this small reduced instance. Similarly, in the Independent Set condition, after all D1 selections have been made, the remaining reduced instance (shown in Figure 5.3) is small.

This requirement, in conjunction with the cognitive support of the interface, displaying the size of the current candidate solution, should render these graphs solvable exactly if the D1 (and in the case of the Random graphs, D2) strategies are acquired, and applied correctly. Further, if these strategies are acquired and applied correctly, very little backtracking should be required to find an exact solution. If a D1 schema is activated, better solution quality should also be accompanied by little to no backtracking. In fact, all graphs in this group should be solvable with no more than a few backtracking steps, should a D1 schema be activated.

The relatively high percent optimal solutions on the D1 graphs supports the hypothesis that participants were able to acquire and apply this strategy, although not consistently. While there was not strong evidence for an interaction between Graph Type and Problem Version, it is still notable that performance on D1 graphs is better, in terms of percent optimal solutions, in the Search condition than the Optimization version. This makes sense if subjects in the Search condition, given the specified goal, are more able to determine that the D1 strategy is correct and apply it more often,

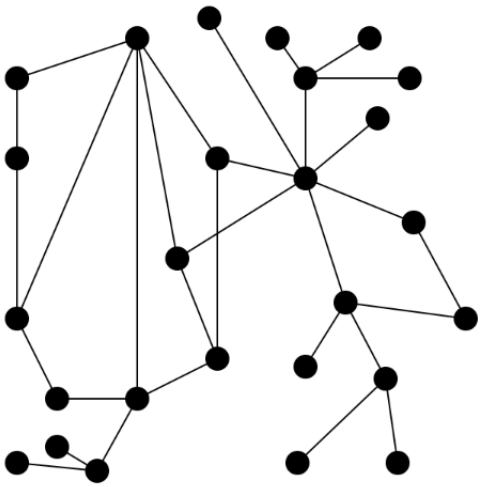


Figure 5.1: Graph 3 before any vertices have been added to the solution

which would result in more optimal solutions.

Correct application of the D1 strategy should result in fewer backtracking moves on graphs to which the strategy is applicable. The number of adjusted undos is lower on D1 graphs than on any other graph types, which also suggests that subjects are able to acquire and apply the D1 strategy to some extent. However, the mean number of unadjusted undos is higher in the Search condition than the Optimization condition, which contradicts the prediction that the specified goal should make, ascertaining the correctness of the strategy easier.

Another measure of D1 strategy acquisition and application is the number of selections made that could be D1 selections. While it is possible to attempt to determine the frequency of D1 moves, without anything to compare the frequency to, this measure alone is not sufficient to determine if the strategy is being applied. What can be compared, however, is the frequency of possible D1 moves between the Search and Optimization conditions. If participants in the Search condition are able to determine, given the specified goal, that the D1 strategy is correct, it would expected that more D1 selections are made in the Search condition than the Optimization condition. However, the data does not support this hypothesis and more D1 at Onset and D1 selections are made in the Optimization condition than the Search condition.

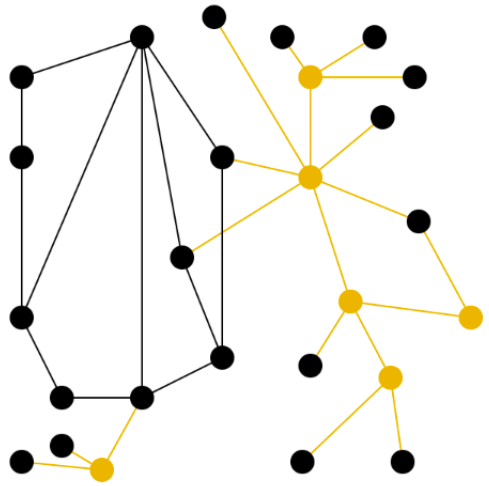


Figure 5.2: Graph 3 after all available Vertex Cover D1 selections have been made. Remaining reduced instance (black edges are uncovered) is small.

In summary, despite the fact that in debrief many participants identified the D1 strategy as one of those they used, it is not clear from the results that participants are able to acquire and apply the D1 strategy with consistency. Further, schema activation is not taking place, in general. While performance is better on D1 graphs than any others, the amount of backtracking and the frequency of possible D1 selections contradict these results. One possible interpretation of this data is that participants are applying the D1 strategy, but not consistently, and other Move selections are interfering with the correct application of the D1 strategy. This implies that the specified goal of the Search condition is not sufficient feedback, in general, for determining the correctness of the strategy with certainty.

Performance as measured by percent optimal solutions on D2 graphs is good, bested only by that on D1 graphs, which suggests that the D2 strategy might be acquired and applied. Unlike D1 selections, D2 selections are less available and also more complex, requiring attention to at least 4 vertices (the three in the triangle, and at least one connected to a member of the triangle). In addition, D2 graphs require both the D2 and D1 strategies, plus some exhaustive search, to be solved. For the graph in Figure 5.4, the state of the graph after all D2 selections have been made is shown in Figure 5.5 for the Vertex Cover Condition, and 5.7 of the Independent Set

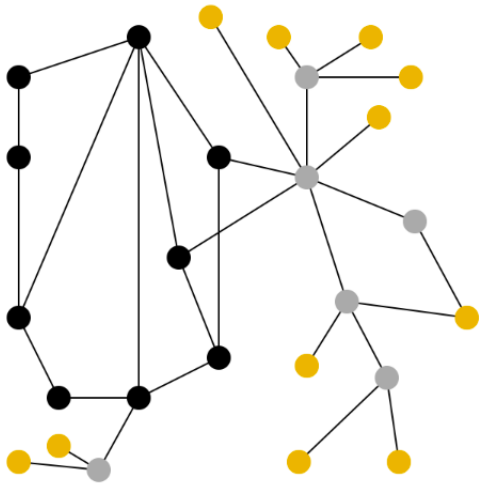


Figure 5.3: Graph 3 after all available Independent Set D1 selections have been made. Remaining reduced instance (black vertices are independent of those added thus far) is small.

condition. The state of the graph after all D1 and D2 selections have been made is shown in Figure 5.6 for the Vertex Cover Condition, and 5.8 of the Independent Set condition.

Better performance on these graphs in the Search condition than in the Optimization condition suggests that feedback from the specified goal may improve acquisition of this strategy. This trend is more apparent in the Independent Set condition, which suggests that it may be easier to acquire and apply this strategy on this problem. An alternative explanation, however, is that these selections are actually due to a Greedy strategy, as these choices are also very low degree.

The number of D2 selections is harder to interpret, as there were far fewer of these types of selections available. In the Vertex Cover condition, D2 at Onset selections are made with high frequency on both D2 and Random graphs. In contrast, very few other D2 selections were made by this group. Interestingly, in the Independent Set condition many D2 and D2 at Onset selections were made on D2 graphs and few were made on Random graphs.

Correct application of the D2 strategy should result in fewer backtracking moves on graphs to which the strategy is applicable. The number of adjusted undos is high

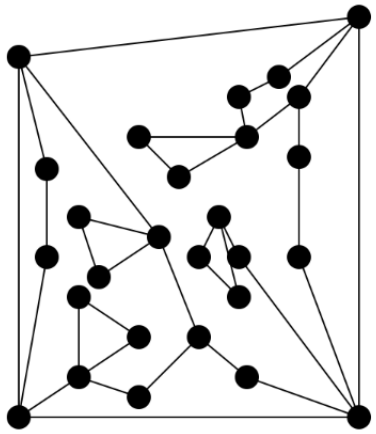


Figure 5.4: Graph 7 before any vertices have been added to the solution

on D2 graphs, surpassed only by that on Random graphs, if at all. This suggests that subjects are not able to acquire and apply the D2 strategy consistently. The mean number of unadjusted undos is higher in the Search condition than in the Optimization condition, contradicting the prediction that the specified goal should make ascertaining the correctness of the strategy easier. Clearly, schema activation is not occurring due to the great number of moves needed to find solutions to these graphs.

In summary, despite the fact that in debrief some participants identified the D2 strategy as one of those they used, it is clear from the results that participants are not able to acquire and apply the D2 strategy consistently, and that schema activation is not taking place. While performance is good on D2 graphs, the amount of backtracking and the frequency of possible D2 selections contradicts these results. Perhaps participants are applying the D2 strategy, but not consistently, and other Move selections are interfering with the correct application of the D2 strategy. Another explanation is that the inconsistent application of the D1 strategy may confound these results as well. As with the D1 strategy, this implies that the specified goal of the Search condition is not sufficient feedback, in general, for determining the correctness of the strategy with certainty.

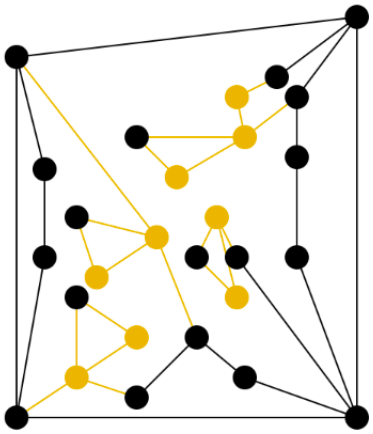


Figure 5.5: Graph 7 after all available Vertex Cover D2 selections have been made.

5.6.2 Heuristic Strategies

Determining with any certainty that the Greedy strategy is responsible for participants' selections is confounded by a number of factors. A vertex could be selected as a greedy choice, even though it is not necessarily the most greedy choice, and therefore determining what constitutes a greedy choice is not easy. Seemingly greedy choices can be explained by other strategies: D2, D1, or local selections. However, it is very likely that participants use greedy moves, they are akin to other known strategies, like hill climbing, applied by novices, and therefore it is assumed to be occurring. Greedy choices, both high degree choices in the Vertex Cover problem, and low degree choices in the Independent Set problem, are relatively easy to make, requiring only visually considering the number of edges connected to a vertex, a local process which can likely leverage the perceptual system.

The aim in designing the graphs in the Greedy Resistant block is to see if participants are able to avoid greedy choices when these choices are sub-optimal. Graphs were designed to be resistant to the heuristic strategy identified in order to evaluate whether or not avoiding the strategy could be observed. The greedy strategy is likely a natural one, frequently identified by participants in verbal debriefs.

The greedy strategy can be successful on all given graphs other than the Greedy

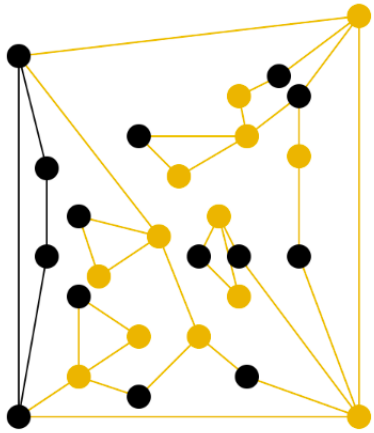


Figure 5.6: Graph 7 after all available Vertex Cover D1 and D2 selections have been made. Small reduced instance (black edges) remains.

Resistant graphs. The strategy is guaranteed to fail, in the Search version of the Vertex Cover problem at least, on all Greedy Resistant graphs. There are three possibilities: the greedy strategy is not acquired and applied; strategy can be acquired and applied consistently, equally across all graphs; or the strategy is acquired and applied, and avoided on Greedy Resistant graphs. If the strategy is not acquired and applied it would expected that relatively low degree choices be made on all instances and possibly good quality solutions on all graphs. If the strategy is acquired and applied equally across all graphs, poorer performance is expected on Greedy Resistant graphs and high degree choices on all graphs. The final possibility is that the strategy is acquired and applied, and avoided on Greedy Resistant graphs. In this case, high degree choices are expected on all instances except those on which the greedy strategy does not apply, with a high percent of optimal vertex covers on all graphs.

The mean degree of choices in both the Search and Optimization versions differs very little on the five graph types. However, it is significantly lower on Greedy Resistant graphs in the Search condition. Performance on Greedy Resistant graphs in all conditions is the worst, as measured by percent optimal solutions, a pattern that holds if the Search and Optimization conditions are considered separately. These findings suggest that if the greedy strategy is acquired and applied, then it is not

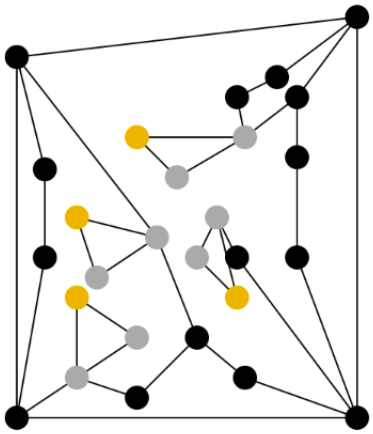


Figure 5.7: Graph 7 after all available Independent Set D2 selections have been made.

consistently avoided on those graphs to which it is designed to fail. The failure to reject this heuristic strategy in the Optimization version could be explained by a failure to recognize that this strategy is sub-optimal. However, it is also possible that the goal modification that is taking place makes avoiding greedy choices unnecessary. Greedy strategy avoidance is not necessary for a minimal modification, or for finding any vertex cover. It may be applicable, depending on how a local modification is implemented. Finally, it is applicable for the specific modification proposed. These results suggest that the specified goal of the Search version is not sufficient to make determining that this strategy is not always correct.

5.7 Impact of Instance Selection and Order

5.7.1 General Impact on Performance

It has been demonstrated in problem-solving research that instance can impact performance as discussed in Section 2.3.2. In this study, graphs were purposely designed to be vulnerable to correct strategies, resistant to a heuristic strategy, and to not be vulnerable to any correct strategy. Randomly generated graphs were also used.

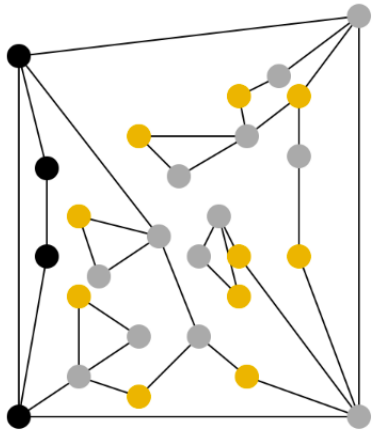


Figure 5.8: Graph 7 after all available Independent Set D1 and D2 selections have been made. Small reduced instance remains (black vertices)

While the acquisition and application of the identified strategies is not conclusively supported by the findings thus far, it appears that participants may be able to acquire and apply these strategies, or else that some other strategies are being applied. Strong evidence was found for differences in performance based on Graph Type on: percent optimal, time per move, moves, toggles, and undos. These results, however, are mixed.

In terms of performance when all conditions are considered together, Greedy Resistant graphs show the worst performance, and D1 graphs clearly show the best performance, as seen in Table A.20. While D1 graphs still show best performance in terms of the amount of search and cognitive load, no clear pattern emerges from the data concerning the other graph types. The specificity of the goal interacts with graph type on: time per move, number of toggles, and the number of undos. Best performance on all measures, and on both versions of the problem is again on D1 graphs. Patterns for the remaining graph type are hard to identify, as seen in Table A.21. Problem also interacts with graph type on: PAO, time per move, number of toggles, moves and undos. Again, best performance on all measures is on D1 graphs, and the remaining results show no clear pattern aside from the fact all (D2, GR, NS, and Rnd) appear harder than D1 graphs. See Table A.22 for details.

What is clear is that of the graph types used in this study, D1 graphs were easiest. No conclusive pattern emerges about the relative difficulty of the remaining graph types.

5.7.2 Impact of Order on Performance

Just as instance type can impact problem solving, the order in which different kinds of instances are presented has also been known to impact performance. In this study, the order of graph type presentation strongly impacted performance, on every measure. Some groups show more variation while other groups show less on the same measure. For example, group A shows much less variation on the percent optimal solutions found than the other groups. On other measures, some groups show overall much poorer performance than other groups. Group B, for instance, has much higher PAO, than the other groups, on all graphs other than the Random graphs, .

If each problem is considered separately, the order of graph type also strongly impacts performance on all performance measures except time per move. In addition to the performance differences between the Vertex Cover and Independent Set conditions, the same kind of pattern emerges, with differences in the amount of variation between problem ordering. Problem order seems to lead to more variation differences in some groups in the Vertex Cover condition than in the Independent Set condition.

A similar pattern emerges if each Problem Version is considered separately; the order of graph type also strongly impacts performance on all performance measures except time per move and the number of undos. The specificity of the goal and resultant problem modification appearsto interact with how graph type ordering impacts performance.

In the next two sections, a deeper analysis of these findings is presented, with the goal to unravel how instance ordering, problem, goal modification, and strategy acquisition all contribute to performance differences as a result of graph type ordering.

5.7.3 Positive Impact on Performance

Acquiring and applying correct strategies, like those based on reduction rules introduced earlier, early on in a series of instances of a problem, should result in improved performance on later instances of the same problem so long as other strategies do not interfere with their correct application. In the graphs used in this study, two blocks of graphs were purposely designed to be vulnerable to the correct D1 and D2

strategies and could be solved optimally using these strategies in conjunction with a small amount of exhaustive search. Early presentation of these graphs should result in improved performance on all subsequent graphs, if the strategies are acquired and applied consistently. However, in the Vertex Cover group, there is potential interference with Greedy Resistant graphs, which are also not particularly vulnerable to either D1 or D2 strategy. Therefore, performance on Random instances only is considered, as they are always presented after D1 and D2 graphs, and they contain D1 and D2 selections.

Group A was presented D1 graphs earliest, in block 1, which gave participants in this group the most opportunity to acquire the D1 strategy, and the potential to activate a schema for it. In both the Vertex Cover and Independent Set groups, this group shows the best performance on Random graphs, in terms of Percent Optimal solutions. Group C was presented D1 in block 2 which gave these participants a good opportunity to acquire the D1 strategy, and the potential to activate a schema for it. As a result, it is predicted that performance should be relatively good on Random graphs; however, this pattern does not emerge from the data. In the Independent Set condition, performance on Random graphs is good, but in the Vertex Cover condition, performance is poor. Group B was presented D1 graphs in block 3, and this late presentation of D1 graphs provided little opportunity to acquire this strategy or activate an associated schema. This was predicted to result in poor performance on Random graphs. In the Vertex Cover condition, this group shows the worst performance on Random graphs; however, in the Independent Set condition, performance on Random graphs was very good. Group D were presented D1 graphs last, in block 4, and had the least opportunity to acquire this strategy or activate an associated schema. In the Independent Set condition, performance on Random graphs is poorest, as predicted. However, in the Vertex Cover condition, this pattern fails to emerge, with second best performance on Random graphs between all groups.

If schema activation for these strategies occurs, it should result in less search, requiring less backtracking, and a reduced cognitive load. Results of this study, however, do not support the theory that schema activation for strategies occurs. Based on this analysis, only Group A is a viable candidate for schema acquisition. However, this group used the most Toggles and Undos on Random graphs in both the Vertex Cover and Independent Set conditions.

These mixed results are challenging to interpret. Group A's performance supports the hypothesis that early presentation of graphs vulnerable to this correct strategy

allowed for the acquisition and application of the D1 strategy, resulting in good performance on Random graphs presented at the end of the problem set. No schema activation appears to occur for this group, however. The remaining block orderings do not support the hypothesis, unless only the Independent Set condition is considered. This observation, coupled with the earlier observation of better performance overall in the Independent Set condition, is interesting. One possible explanation is that the relative lack of Greedy Resistant graphs in the Independent Set condition either makes acquisition of this strategy easier, or alternatively, that acquisition of the Greedy strategy is not discernible from the acquisition of the D1 strategy.

Results for the Independent Set condition differ from that on the Vertex Cover condition. Group A, with earliest introduction to graphs vulnerable to these correct strategies, shows better and best performance on subsequent graphs in terms of Percent Optimal, but poor in terms of PAO. See Figure 5.9. Groups B and C, with mid-way introduction to these graphs, show good performance in terms of Percent Optimal, but Group B shows poor performance in terms of PAO, whereas Group C has good performance on this measure. Finally, group D, with the latest introduction to these graphs, has worst performance on Random graphs as measured by Percent Optimal, but good, as measured by PAO. These seemingly contradictory results are likely due to the relatively poor performance on this problem in the Optimization condition.

When measures of search and cognitive load are considered, results for the Independent Set condition are mixed. Early introduction of D1 and D2 vulnerable graphs in Group A had mixed impact on both the number of Toggles and the amount of backtracking. Mid-way introduction has equally mixed impact on the cognitive load and backtracking needed in Groups B and C. Group B uses fewest toggles and Undos on Random and Greedy Resistant graphs, but many Undos and Toggles on NS graphs. Group C uses a moderate amount of toggles and undos on graphs following D1 and D2 graphs. Group D, with latest introduction to these graphs, uses many toggles, and performs a lot of backtracking. Clearly, schema acquisition is not occurring for any graph ordering on this problem. Details are found in Figures 5.10 and 5.11.

These results differ from those on the Vertex Cover condition. Early introduction, in the Independent Set condition, to graphs vulnerable to these known correct strategies has a less pronounced positive impact on performance and also appears to result in more cognitive load and backtracking. Mid-way introduction to these D1 and D2 vulnerable graphs shows good performance on subsequent graphs, and also

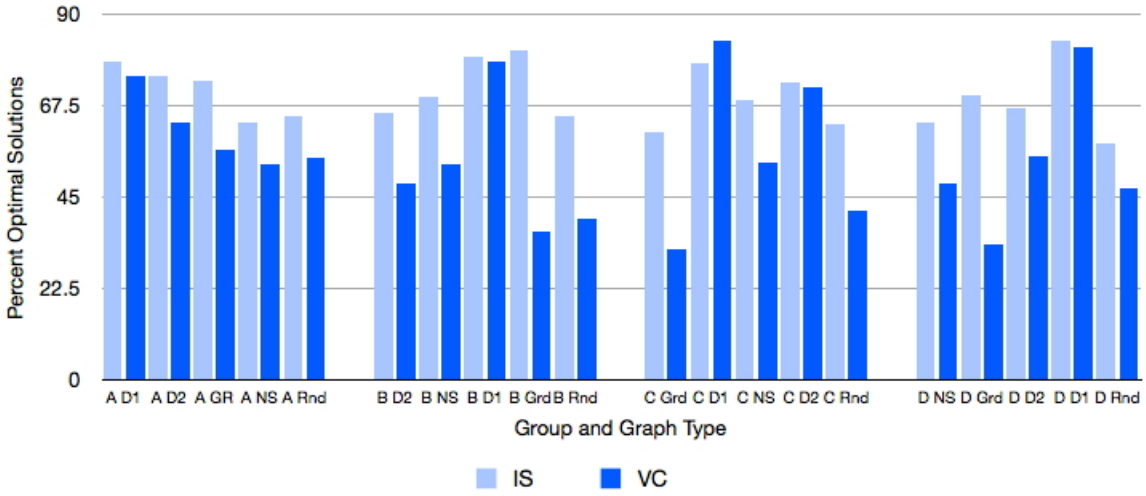


Figure 5.9: Percent Optimal Solutions in Order Presented by Graph Type

little search and reduced cognitive load in one group but poor performance and some increased cognitive load and backtracking in another. Finally, late introduction to these graphs has mixed impact on performance and tends to increase the amount of backtracking and cognitive load.

Lack of interference with Greedy Resistant graphs might explain this. Lacking Greedy Resistant graphs, participants in the IS condition may end up relying upon the Greedy strategy much more. Further, this strategy is indistinguishable from D1 strategy, as if there is a D1 move available, it must also be among the lowest degree vertices in the graph.

5.7.4 Negative Impact on Performance

A simple strategy like the Greedy strategy is predicted to be likely adopted by problem solvers on problems like these. It can be successful on many of the graphs presented, and this success can lead to reliance upon it. Confronted with graphs to which this strategy is resistant could conflict with this previous success. The more established the strategy, likely the harder it is to resist, which could manifest in increased backtracking, and worse performance on Greedy Resistant graphs.

Group A was given Greedy Resistant graphs in Block 3, and therefore should have had a fair amount of time to acquire the Greedy strategy before facing graphs to which it is resistant. Contrary to what is predicted, however, they show the best performance

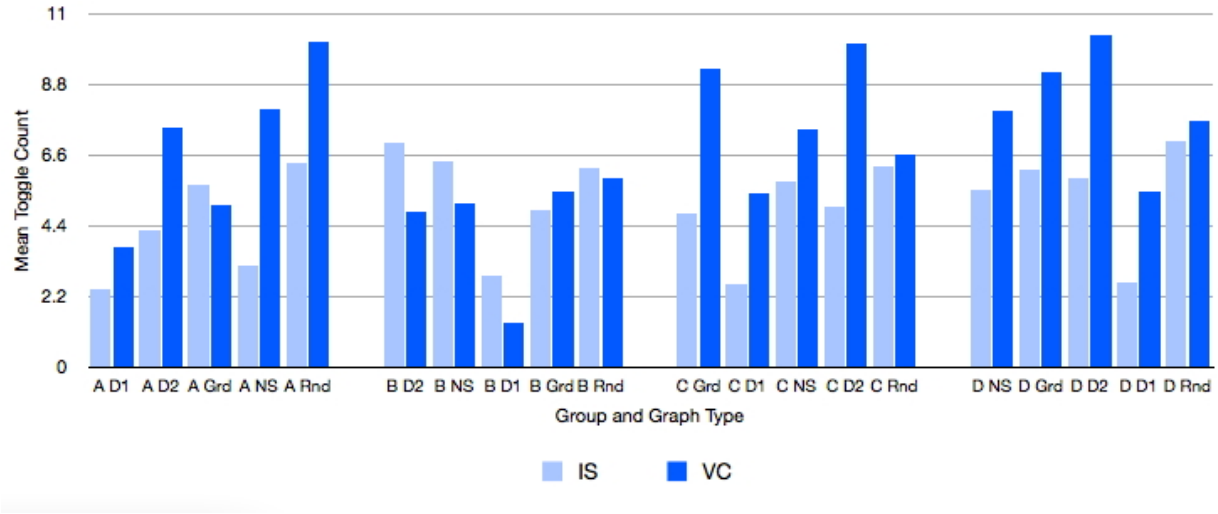


Figure 5.10: Toggles in Order Presented by Graph Type

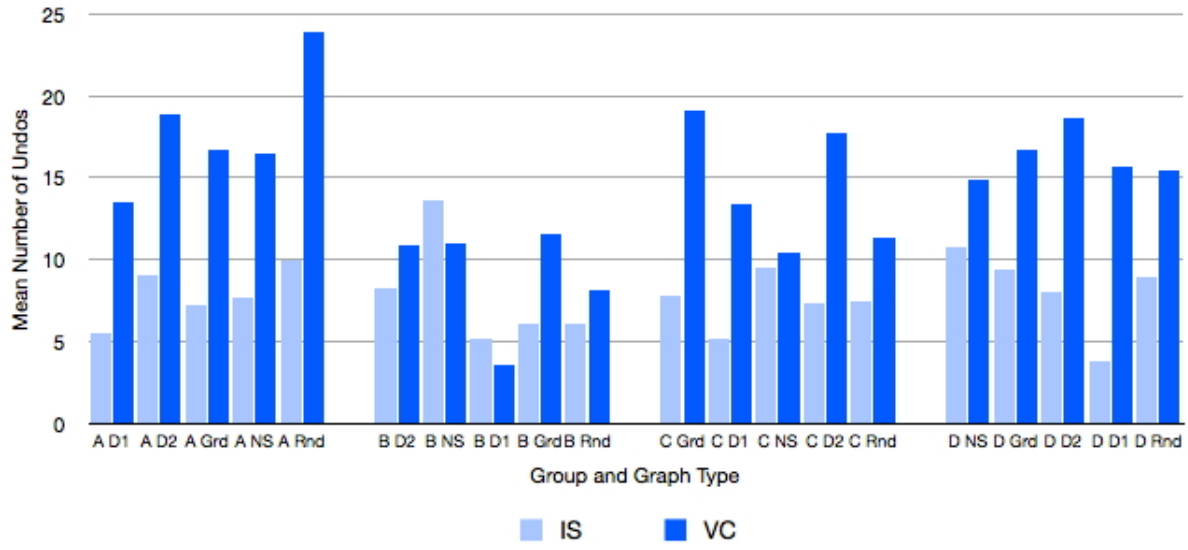


Figure 5.11: Undos in Order Presented by Graph Type

on Greedy Resistant graphs. This seeming contradiction could be explained, however, by the relatively large amount of backtracking done by this group. While not the highest in the study, it is still significant. Group B was given Greedy Resistant graphs in Block 4, and should therefore had the most time to establish the Greedy strategy as successful. However, performance on these Greedy Resistant graphs is bested only by that in Group A, which contradicts the prediction. Interestingly, this group also uses the least backtracking. So it appears that this group, with plenty of opportunity to establish the Greedy strategy as successful, was able to quickly and easily reject it on graphs on which it failed. Group C received Greedy Resistant graphs in the first block, before having any opportunity to acquire the Greedy strategy. Again, in contradiction to the prediction, performance in this group to Greedy Resistant graphs is the worst of all graph orderings. Coupled with the fact that they also used the most backtracking on these graphs, this indicates that despite having little opportunity to acquire the Greedy strategy, this group struggled with these Greedy Resistant graphs and required a lot of searching to find these poor solutions. Group D was given Greedy Resistant graphs in block 2, and had little opportunity to acquire the Greedy strategy. However, this group shows second poorest performance on these Greedy Resistant graphs, and a great deal of backtracking. Like group C, this group with little opportunity to establish the Greedy strategy, required a great deal of search to fare relatively poorly on the Greedy Resistant graphs.

It is clear from these results that, if the Greedy strategy is indeed being acquired, rejecting it when it cannot succeed is not impacted by the number of graphs encountered beforehand upon which it can succeed. One possible explanation for these findings is that this strategy is not actually consistently acquired and applied by participants. However, this contradicts verbal reports of its use. Another explanation is that despite its acquisition, it is easily replaced by other strategies when it fails. In particular, Groups A and B, who had the most time to acquire this strategy, equally had a lot to time to acquire other strategies, and therefore may be best equipped to alter their strategies upon realization that the Greedy strategy is not applicable. Both these groups had encountered D1 and D2 graphs prior to the Greedy Resistant graphs, which could have provided them with viable alternate strategies, or at least an awareness of the existence of multiple strategies. Groups C and D, on the other hand, with the least experience on other graphs, and in particular none to which simple correct strategies were applicable, had the least opportunity to acquire correct strategies. This lack of flexibility may have impeded their ability to switch strategies,

leading to poor performance with a great deal of exhaustive search.

5.8 Summary

Performance on the Optimization version of both the Vertex Cover and Independent Set problem is consistent with previously reported performance results on other hard Optimization problems, namely E-TSP. This finding supports Hypothesis 1, and suggests that problem-solving performance on this kind of hard optimization problem may not be solely dependent on visual processes that are only applicable to Euclidean Problems. It is possible that problem solving on these not-Euclidean may still leverage, at some level, powerful visual processing mechanisms. But, it is not immediately clear how this might manifest itself.

Performance on the Search version of both the Vertex Cover and Independent Set problem is significantly better than that on the associated Optimization version, a finding that supports Hypothesis 3. Cognitive load was also found to be higher in the Search condition that in the Optimization version, in support of Hypothesis 5. Participants in the Search condition used far more search to find their solutions, found optimal solutions with higher frequency, and showed signs of greater cognitive load, all indications that the problem being encoded in the Search condition differed from that being encoded in the Optimization condition. This in turn implies that when tasked with the Optimization version of a hard computational problem, and facing a goal that is not encodable, problem solvers modify the problem and encode some other goal. This has important implications, not only in this work, but also in the interpretation of the results of previous studies which used hard Optimization problems as instruments. How might performance on other hard problems differ if participants were tasked with the Search or Decision version? In this work, performance was found to be better, and it is interesting to wonder if this might also be the case on hard problems like E-TSP.

Another important implication of these results relates to persistence and problem solving. In education, life, and work, people face many problems. Polya suggested that encouraging engagement with problem-solving tasks is important in order to develop good problem-solving skill. The significantly greater search observed in the Search condition of this study indicates that participants in this condition are much more persistent than those in the Optimization condition, that they are more engaged with the problem, despite working on a harder problem, under heavier cognitive load.

The careful specification of goals may have educational implications, in particular in problem-solving education. In addition, while previous work concluded that goal specificity impeded problem solving, the results of this study contradict this claim.

A framework for identifying candidate problem modifications was proposed, and based on this, four problem modifications were identified to explain what problem solvers tasked with the Optimization version of these problems might be solving. In the case of the Minimum Vertex Cover problem, the greatest support was found for the Minimal Modification, suggesting that the non-encodable aspect, that is the optimality of the vertex cover, is replaced by the easier to evaluate goal of finding a minimal vertex cover. Similar results were found for the Maximum Independent Set problem, with the Maximal Independent Set problem best matching the results. This finding supports Hypothesis 4, although caution must be taken in interpreting these results. Four candidate modifications were proposed in this work and evaluated in terms of a number of performance, search and cognitive load measures. While by all accounts the Maximal/Minimal modification is the best fit for the results among those modifications identified, there are possibly many explanations for this match. The goal of either or both the given problems could have been encoded such that a different modification occurred, other than any identified here, for which maximal/minimal solutions are coincidentally likely.

The implications of this finding, that goal modification occurs when problem solvers are tasked with hard Optimization problems, are non-trivial. What kinds of modifications might explain performance on other hard Optimization problems, and how might that impact the interpretation of previous human performance results on hard optimization problems?

Modification of a problem with an ill-defined goal was predicted to result in a problem that is easier, one which results in lower cognitive load. The results of this study strongly support Hypothesis 5. Whatever goal modification(s) are taking place, result in a problem-solving task which is significantly easier than the corresponding Search version.

Differences emerged in performance between the two problems used in the study. In contradiction with Hypothesis 2, performance was significantly better on the Independent Set problem than on the Vertex Cover problem. These results are somewhat counter intuitive. The Vertex Cover problem, being both a minimization problem and fixed parameter tractable was hypothesized to be easier, potentially putting less strain on the cognitive system. Surprisingly, this was not the case. There are a number of

possible explanations for this finding. If participants were able to consistently acquire and apply the reduction rules identified here, good performance on the Independent Set problem might be explained, because although the k-Independent Set problem is not known to be fixed parameter tractable, the strategies identified here, and used to generate a good portion of the graphs, are applicable to both versions of both problems. However, this would still not be sufficient to explain the better performance, only equally good performance. Another confounding factor, is that the inclusion of graphs which were resistant to the Greedy Strategy only in the Vertex Cover condition, could have impacted performance overall. Performance on the Independent Set problem is still better when these Greedy Resistant graphs are excluded from the analysis, implying that if this is the source of the difference, then these Greedy Resistant graphs impacted performance on other graphs as well in the Vertex Cover condition. If this is the case, then this speaks to how sensitive performance results can be to instance selection. However, it may also be that either instances of the Independent Set problem are subjectively easier to solve than instances of the Vertex Cover problem, or that maximization problems may not be harder than minimization.

While the problems used in this study are, in general, computationally hard, there still exist many instances for which simple correct strategies can be applied that can either solve the instances exactly, or at least reduce them to a small manageable reduced instance. Ten such graphs were included in this work to test whether or not simple correct strategies could be acquired and applied. Results of this study are not conclusive, and do not strongly support Hypothesis 8, despite the fact that many participants verbally described using the simplest correct strategy identified here. Further, the specified goal of the Search condition was not sufficient to reinforce the correctness of the two identified correct strategies.

Similarly, while it was assumed that participants would be likely to apply the heuristic Greedy strategy, the results of this study do not strongly support this assumption, nor do they support Hypothesis 10. The specified goal of the Search condition was also not sufficient to reinforce the incorrectness of this heuristic strategy on Greedy Resistant instances.

Despite the lack of support for the acquisition and application of the D1 strategy, performance on graphs designed to be vulnerable to this strategy was significantly better than on any other type of graph. As proposed in Hypothesis 6, instance type can have a significant impact on performance. However, if the the acquisition and application of the D1 strategy is not the explanation for the better quality on these

instances, it is not clear what is.

The order in which graphs were presented impacted performance, but in a way that is challenging to interpret. The results of the study support Hypothesis 7. However, no clear pattern emerges from this data, and as a result there is no support for either Hypothesis 9, or Hypothesis 11. It is important to note that this study was not designed to specifically investigate strategy acquisition as discussed and analyzed here. Graph types were chosen to test how instance properties impacted performance, to try to determine if D1 and D2 strategies could be applied, and if Greedy choices could be avoided. The analysis done here takes this one step further, to see how the order of instances impacts the acquisition of these strategies. Clearly there are too many confounding factors in this design to draw any conclusions about how graph ordering impacts performance, only that it does.

Chapter 6

Modelling Human Performance on the Vertex Cover Problem

Human performance on the Search and Optimization versions of the Vertex Cover problem indicates that people are able to find optimal solutions, or close to optimal solutions, frequently on instances of this problem, despite the fact that it is computationally hard, a finding that is in alignment with previous work on human performance on other hard computational problems. Heuristics based on simple strategies have been proposed to explain performance on E-TSP, including a hierarchical model using clustering [32], nearest neighbour [25], cheapest insertion [56], crossing avoidance [113], and convex hull, [54]. Due to the Euclidean nature of E-TSP, many of these models work by taking into account both local and global processes. The Vertex Cover problem, in contrast, is not-Euclidean, and global visual processing may be be less important in finding solutions to instances of it. As such, the model proposed here tests the theory that human performance on this problem can be explained using simple, local strategies. In keeping with the hypotheses presented earlier, differences in whether or not the given goal is encodable between the two versions of the Vertex Cover problem appear to result in two different problems being encoded in the internal representation of the problem solvers. This is reflected in how performance is modelled for the different versions.

6.1 The Task

The Vertex Cover problem asks for a set of vertices for a given graph that covers all edges in the graph, called a vertex cover, of either minimum size, resulting in the optimization version, or being no larger than a given k value, resulting in the search version of the problem. Both these versions of the Vertex Cover problem are computationally hard. This problem can also be classified as belonging to the set of problems of transformation (although goal of the optimization version is ill-defined, as covered previously in Section 2.1 above) and therefore the process of finding a solution to an instance can be thought of as finding a sequence of moves that leads to the goal state.

Solving an instance of this problem, by determining a path to the goal, or equivalently a sequence of moves that results in the goal, can require a great number of steps. Due to the computational complexity of the problem, there likely does not exist a general purpose strategy which will always yield the solution and human participants may resort to general problem-solving strategies; however, it is also possible that participants will identify either correct or heuristic strategies that can successfully result in valid solutions on some instances or assist in reducing instances to smaller kernels.

6.1.1 Generalized Performance Results

As shown in Chapter 4, participant performance differs significantly between the search and optimization conditions of the Vertex Cover problem. Since the instances given to participants in the Search and Optimization versions differed only in the statement of the goal, this difference is likely due to the differences in well-definedness of the given goal. Three measures of performance are considered here: the Δ-Opt, PAO, the number of touches (or moves) made to find a solution[1], and the percent of solutions which are optimal. On all measures, significant differences were observed between the conditions. Despite these differences, there also appear to be similarities in how participants choose moves. In both conditions, participants' solutions were frequently minimal covers, and most participants found some optimal solutions to the given instances. Additionally, in both groups, participants spent a short amount of

[1]These models were implemented prior to the main analysis, and as a result, while in the main analysis the moves made were adjusted so that they did not include Toggles, in this Chapter, they are not. This toggling behaviour was discovered as a result of close inspection of participants' choices when attempting to explain the model fit, as discussed below.

time per vertex finding a solution (roughly 2.5 seconds per move), with no significant difference between these groups, which is in keeping with a fast and efficient move selection process, and little or no sub-goaling, planning of sequences of moves, or multistep move selections.

One of the challenges of comparing performance between the Search and Optimization conditions is that participants in the Search condition were able to give up if unable to find a solution that matched the goal. As a result, in previous sections, measures of the relative solution quality as compared to optimal were not considered for participants in the Search condition. For this analysis, since only a small number of participants is considered, it was possible to select them such that all solutions, independent of their optimality, were valid. This allows for the comparison of relative performance in both conditions. However, it also means that when considering the Search condition as a whole, mean values for the PAO or $\Delta - Opt$ cannot be given.

Optimization Version

In the optimization version, performance was characterized by mean PAO of roughly 10%, and fewer than 50% optimal solutions. Participants spent little time or effort searching for a solution, with just over two seconds spent per vertex selection, on average. This is good indication that moves were likely based on local information available at the current state, with little information from previous states. As a consequence, vertices selected tend to be close to the most recent previous selection, measured either as path length, or Euclidean distance. The number of moves used to find a solution could be indicative of an overall strategy that involves some level of local optimization, but lacking consistent global optimization. These results are summarized in Table 6.1.

Search Version

In the search condition, performance was characterized many optimal solutions. Participants spent little time selecting vertices, but with a great number of moves to find a solution. The time per vertex selected indicates that, like the Optimization group, it is likely that simple strategies, based mainly on local information, is responsible for vertex choices. However, the number of moves leading to a solution indicates that there is also some global optimization strategy at play, or a general strategy like trial and error. These results are summarized in Table 6.1.

Table 6.1: Summary of mean performance differences on Vertex Cover problem. Measures of performance are: mean Δ-Opt, mean PAO, mean percentage of optimal solutions (% Optimal), number of moves or selections (Touches), and mean time per vertex selection (Time).

Condition	**Δ-Opt**	**PAO**	**% Optimal**	**Touches**	**Time**
Optimization	1.12	9.597%	41%	24.70	2.34
Search	N/A	N/A	70%	37.89	2.37

6.2 The Vertex Cover Problem Models

The models developed in this work for the Vertex Cover problem assume that the Optimization and Search versions involve solving different problems. As such, the models for these two versions of the Vertex Cover problem are considered separately.

6.2.1 The Vertex Cover Problem Model I

The model for the Optimization version of the problem worked by finding a cover based on local choices, without trying to find alternate, more optimal covers. This is based on the assumption that the ill-defined goal of this version of the problem renders the search for an optimal solution infeasible. The model's process consisted of two stages: Cover Finding and Optimization. In the Cover Finding stage, the model made vertex selections by probabilistically choosing one of four simple strategies. These strategies were identified based on observations of participant tendencies, and on the correct strategies (as defined previously in Section 3.2.1 above) derived from parameterized complexity theory. In the Optimization stage, local optimization was applied to the valid cover.

The model assumes that subjects only use information about the current state of the instance, which follows from the speed with which subjects make vertex selections. This speed is not in keeping with making multistep move selections, or planning sequences of forward moves. The model also assumes that participants do not maintain or access collections of subgoals. This is in alignment with how naive subjects are conjectured to solve these kinds of problems [70], and with the constraints problem-solving processes place on storage in working memory [9].

Cover Finding Stage

The model applied four local strategies probabilistically to find a vertex cover. These strategies can be divided into two groups: correct and heuristic.

Only one correct strategy was used, the D1 strategy. The D1 strategy, as described earlier in Chapter 3, involves adding vertices connected to vertices of Degree 1. This is a simple strategy, which was reported by many participants as a strategy used. The D2 strategy, while also possibly appropriate, was not included in the model as less support for its use was found. Analysis supports the hypothesis that it is unlikely a dominant strategy.

Three heuristic strategies were used by the model: the Greedy strategy, the Close strategy, and the Redundancy Avoidance strategy. The Greedy strategy works by choosing high-degree vertices to add to the cover. Participants were observed applying this strategy and frequently reported having made use of it during the problem-solving task. It was implemented in the model by identifying the set of highest degree vertices, and selecting one at random. Another frequently observed pattern was participants' tendency to follow a path through the graph, adding alternating vertices to the cover. This amounts to considering nearby (in terms of path length) vertices not directly connected to the most recently added vertex, and adding one that is not already in the cover. The Close strategy was implemented by identifying all vertices not already in the cover at distance 2 from the most recently added vertex, and choosing one at random. Finally, a local strategy was identified to minimize the amount of redundancy in the vertex cover. It worked by avoiding adding vertices to the cover that would *doubly* cover an edge. An edge was considered doubly covered if both vertices to which it is connected are included in the cover. This Redundancy Avoidance strategy was implemented in the model by identifying the set of vertices in the graph not yet in the cover, with no neighbours in the cover. One vertex was selected from this set, at random.

At each step, the model selected one of these four strategies probabilistically, and as a result the model had four associated parameters, one for each strategy. Each parameter representing the probability of selection.

Let S_i be the parameter value for Strategy i, where $i \in \{D1, Gr, Cl, RA\}$, representing the four identified strategies: D1, Greedy Strategy (Gr), Close (Cl), and Redundancy Avoidance (RA), respectively. The values of each S_i are elements of $\{0.0, 0.1, 0.2, ..., 1.0\}$. Thus, the formula to represent valid parameter values is:

$$S_{D1} + S_{Gr} + S_{Cl} + S_{RA} = 1.0 \tag{6.1}$$

If the model chose a strategy that is not applicable, a vertex was selected at random from the set of vertices not yet in the cover. This was added to ensure that the model would always terminate. For instance, if the D1 strategy was selected, and no D1 vertices were available, the model would select a vertex at random that is not yet in the cover. Once a valid cover was found, the model switched to the Optimization Stage.

Optimization Stage

While the model assumed that global optimization is infeasible, local optimization was assumed to be feasible, and was applied to the valid cover. Once a valid cover was found, the model attempted to locally optimize the cover by searching for vertices in the cover that could be removed from the cover without rendering the cover invalid. The model terminated once a valid cover is found for which no more local optimization is possible.

Rationale

The scope of this model was limited to the behaviour of subjects who are novices at solving this kind of visually presented hard computational problem. As such, it assumed that subjects do not have available to them complex or powerful move selection strategies that might be learned through experience. Due to the computational complexity of this problem, such an assumption might even hold for non-novices, because of the number of possible paths to the goal which must be considered for more complex strategies. This model also assumed that the infeasible-to-identify goal of the given task was not encoded, and therefore the model made no attempt to determine whether or not a candidate solution was optimal. The model assumed that subjects have a predetermined disposition to apply strategies with specific likelihoods, a simplifying assumption that, while perhaps not representative of the precise mechanism by which subjects select vertices, was still reasonable, as subjects have no meaningful mechanism by which to compare the success of these strategies. Because subjects were not able to feasibly determine when a solution was optimal, nor even reasonably determine if a more optimal solution existed, they were likely unable to make associations between strategies and relative success that would lead to alterations in the

likelihood of a strategy being applied on latter instances.

Each move selection was independent; that is, it did not depend on the relative success or failure of any previous moves. This simplifying choice is in alignment with the fact that participants in the optimization condition do not have information available to them which would help them learn if a strategy is successful or not.

All four strategies used in the Cover Finding stage, as implemented in the model, have memory requirements which on large instances may exceed memory limitations of the human cognitive system. In some problem states, the set of vertices which is considered may be large. Consider the case where the Redundancy Avoidance strategy is chosen on the first step of a large instance. This would result in all vertices in the graph being added to the set of vertices to be considered. However, the inner mechanism of adding all vertices to the set and selecting one at random is not meant to directly model the inner mechanism implemented in the cognitive system, but rather attempts to simulate how one of many candidate vertices may be selected, as this process is unknown. So while the implemented process may exceed the limits of memory, the model assumes that this mimics, outwardly at least, how a vertex is chosen. The choice to select a vertex at random if a strategy fails was a simplifying assumption, and alternatively another strategy could have been selected instead of a random vertex, and could perhaps improve the performance of the model.

The Cover Finding stage makes up the bulk of the model: finding a valid vertex cover by making locally optimal choices. Importantly, the model will always find a valid vertex cover. In this study, participants were unable to proceed to the next instance in the set before finding a valid cover, an artificial constraint. However, in pilot studies, it was found that participants consistently were able to identify valid covers even without this constraint, and therefore this choice was deemed reasonable.

In the Optimization stage, the model locally optimizes the found cover, by considering each vertex in the cover and, if all of the vertex's neighbours are also in the cover, removing it from the cover. This local optimization process has low memory requirements, as each vertex in the cover can be considered independently.

6.2.2 The Vertex Cover Problem Model II

The model for the Search version of the problem worked by finding a cover, of size not exceeding the k value of the goal, by making local choices. This was based on the assumption that the well-defined goal of this version of the problem renders the

search for an optimal solution feasible. The model used Model I as a core, with two additional parameters: persistence and undo-depth.

The model developed for the Search version of the Vertex Cover problem, like Model I, assumed that subjects only use information about the current state of the instance, in keeping with the speed with which subjects make vertex selections. However, unlike Model I, Model II assumed that participants more persistently search for a solution which matches the goal. This was assumed to manifest as an iterative process of attempting to find a valid cover, comparing its size to the size, k, of the goal, and if its size exceeds k, undoing some number of selections, and trying again.

Persistence and Undo Stage

The model assumed that subjects would try to find a cover by persistently trying to find different covers until either one was found that meets the goal's k value, or they gave up. The model also assumed that this may be accomplished by undoing some proportion of their previous work. The model removed a proportion of vertices from the cover, choosing vertices at random, then returned to the Cover Finding and Optimization stages of Model I.

Rationale

Again, the scope of this model is limited to novice problem solvers who do not have available to them complex or powerful move selection strategies that might be learned through experience. This is supported by the relative speed with which subjects select vertices when tasked with this problem. The model assumes that subjects will backtrack and try to find an alternate cover when the currently identified cover does not match the goal's k-value. The implementation of this mechanism in the model randomly removes a predetermined proportion of vertices from the current cover, and then, based on the set probabilities, will find another valid cover. This mechanism likely does not match how the cognitive system identifies vertices to *undo*; however, the process by which vertices are selected is unknown and likely differs between subjects. Further, the strategy may differ between instances, and even change within one problem-solving session on a single instance. As such, randomly selecting vertices, which operates within the constraints of working memory, was selected as a simplifying assumption.

As a result of using Model I as the core, Model II did not account for any learn-

ing that might take place based on the relative success or failure of the application of a strategy. Strategy acquisition and rejection likely took place with some participants, because the specified goal enables association between strategy application and success or failure.

Model II never added more than k vertices to the cover, a simplifying assumption that may not accurately reflect how participants explore an instance of the problem.

6.3 Results

Participants were chosen from each group to reflect a range in performance and behaviour, as described below. Model I was fit to three individual participants' performance results in the Optimization Condition, and Model II was fit to two individual participants' performance results in the Search Condition. This section presents the specific results of fitting the models to these subjects.

A number of measures are typically used to compare model and subject performance. It is not uncommon when modelling human performance on hard computational problems to evaluate the solution size (standardized, normalized, or as it deviates from optimal), and the frequency of optimal solutions (See, e.g., [32, 52, 112]). Another approach is to compare solution properties (See, e.g., [45, 53, 106]). Finally, the time required to find a solution is compared, either overall or per move (See, e.g., [10, 77, 85]). Interestingly, few studies of human performance on hard computational problems have reported the number of moves needed for either subjects or the model to find a solution, a measure which has the potential to reveal how much search is needed to find a solution.

The three main measures of performance used in this comparison, were Δ-Opt, percent optimal solutions, and mean number of selections used to find a solution. The Δ-Opt was calculated by subtracting the optimal solution size from the model/subject solution size. The percent optimal solutions was calculated by dividing the number of optimal solutions by the total number of instances completed and presenting the result as a percentage. The mean number of selections was calculated by summing the total number of selections used to find a solution on each instance, and dividing this value by the total number of solutions found. While PAO was used in previous sections as a measure of performance, it was not used to compare the model's solution quality to subjects on a instance-by-instance basis. PAOs as a performance measure are dependent on the ratio between the difference in solution size and the size of

the optimal solution. As such, the PAO for instances with small optimal solutions is inflated relative to those instances with large optimal solutions. For this reason the Δ-Opt was used instead to compare solution quality, a choice that is reasonable for this problem for which little variation in solution quality is found.

Vertices selected were analyzed based on their properties in order to infer what strategies were used in the problem-solving process. These included the number of D1 vertices selected, the degree of vertices selected, the distance between selections, and redundancy avoidance selections. The normalized mean D1 selections were calculated by determining for each instance the number of D1 selections divided by the total number of selections, and finding the mean value across all instances. The number of D1 selections was determined by the current state of the graph at the time the vertex was selected, and therefore a vertex was considered a D1 vertex based on the graph formed by the vertices not yet included in the cover, and the edges not yet covered by a vertex in the cover. As such, vertices which were not necessarily D1 vertices at the start state, may have been D1 vertices at another intermediate state. The mean distance was calculated as the path distance between subsequent selections, divided by the total number of selections. The normalized mean degree of selections was calculated as the degree of the vertex, divided by the maximum degree of the graph, divided by the number of selections. Finally, the number of redundancy avoidance selections were counted per instance, normalized by the number of selections, and the mean number of redundancy avoidance moves was calculated across all instances.

Further analysis of the subjects' solutions to the large instances (instances 4, 9, 14, 19, and 24), was done to compare the subject's solutions to those found by the model. The model was run 100 times for each instance and the frequency of the inclusion of each vertex in the 100 solutions was calculated and compared to the vertices which were included in the subject's solutions. A vertex was considered a match to one in the subject's solution if it occurred in at least 60/100 solutions. The number of D1 and high-degree vertices included in the final solution (if appropriate) was likewise compared. A vertex was considered high degree if its degree was equal to the degree of the highest or second highest degree vertex in the instance.

For these large instances, the subjects' selections were further analyzed to identify two distinct behaviours. First, the subject's selections were analyzed to identify long sequences of undo moves. The interface included an undo button, which would sequentially undo selections. This was included to reduce the working memory requirements of undoing long sequences of moves. Second, the subject's selections were

analyzed to identify the toggling on and off of vertices. When a vertex was added to the cover, all adjacent edges not yet covered by a vertex were changed colour to indicate that they were now covered, and the vertex was also changed in colour to indicate it was in the cover. When removed from the cover, the vertex was returned to the default cover and any edges not covered by another vertex were likewise changed in colour. This process provided an easy-to-use visual impact of the inclusion or exclusion of a vertex, essentially *lighting up* the impact of a choice. It was observed by the researcher that some participants would toggle vertices on and off, sometimes repeatedly, and it was hypothesized that this behaviour was indicative of the subject testing what the impact of a vertex's addition or removal from the cover might have, leveraging the cognitive support the tool supplied. As a result, these toggles (either from on to off, or from off to on in subsequent moves) would not be *selections* in the sense that the model selects vertices, but rather indicative of *testing* or *playing* with the current state of the instance. Since the evaluation of the participant's selections was time consuming, it was only performed on the large instances.

6.3.1 Participant Selection

For the Optimization Condition, three participants were chosen whose performance differed on all measures (solution quality, number of selections, and percent of optimal solutions). Based on this criteria, subjects s17, s20, and s84 were selected. For these subjects, the following measures are presented: mean Δ-Opt, percent optimal solutions, mean selections, normalized mean D1 selections, mean distance between selections, and normalized mean degree of selections. See Table 6.2 for details.

Subject s17's performance was characterized by a large proportion of optimal solutions and very-close-to-optimal solutions. The number of selections this subject needed to find a solution was generally close to the size of the solution, likely related to the fact that the subject found an optimal cover in very few selections on many instances. See Table 6.2 for details. Exceptions to this pattern were most large instances, D2 instances, and the randomly generated instances. This participant made many D1 selections, had a high tendency to make high degree choices, had some redundancy avoidance, and made relatively close selections.

Subject s20's performance was characterized by few optimal solutions and solutions which were not close to optimal. This participant notably moved onto the next instance as soon as a vertex cover was found, the number of selections precisely

matching the size of the final solution on every instance. This participant made a large number of D1 selections, tended to make high degree selections, had some redundancy avoidance, and made very close choices. See Table 6.2 for details.

Subject s84's performance was characterized by some optimal solutions and solutions which were somewhat close to optimal. This participant notably made a great number of selections before settling on a solution, on nearly all instances, regardless of size or type. Vertex selections were typically low-degree, included relatively few D1 choices, were very close, and were not redundancy avoidance selections. See Table 6.2 for details.

Table 6.2: Summary performance results for Optimization condition subjects s17, s20, and s84. Mean Δ-Opt was calculated as solution size - optimal solution size. % Opt Sol is the percent of solutions found that were optimal. Mean Selections is the mean number of (non-adjusted) selections used to find a solution. Mean D1 is the mean proportion of selections that were D1 vertices. Mean Distance is the mean path length between subsequent selections. Mean Degree is the mean Degree of selections normalized by the number of selections made. Mean RA is the mean proportion of selections that could be explained as Redundancy Avoidance selections.

Subject	**Mean Δ-Opt**	**% Opt Sol**	**Mean Selections**	**Mean D1**	**Mean Distance**	**Mean Degree**	**Mean RA**
s17	0.4	68.0%	27.08	0.462	1.874	0.627	0.4721
s20	1.52	20.0%	13.08	0.683	2.264	0.595	0.5188
s84	0.92	48.0%	62.56	0.245	1.666	0.574	0.2502

For the Search Condition, two participants were chosen from those with all valid solutions, whose performance differed significantly on all performance measures. For this condition, more subjects were eliminated, as subjects were able to *give up* on an instance if they were not able to find a solution of size less than or equal to the given goal. As a result, if they gave up with an invalid cover, no meaningful comparison was possible between the chosen measures (on that instance) and the measures of the model's solution. Based on these criteria, subjects s5 and s22 were chosen.

Subject s5's performance was characterized by all optimal solutions, and a large number of selections to find a solution. This participant made few D1 selections, selected a fairly high number of high-degree vertices, showed little redundancy avoidance, and made very nearby selections. Most notably, this participant used a great deal of backtracking before finding a solution, with over 200 selections on a number of instances. See Table 6.3 for details.

Subject s22's performance was characterized by few optimal solutions, and solutions which were not close to optimal. This participant performed very little backtracking before finding a solution, and as a result they moved on to the next instance without finding a solution matching the given k-value on a great number of instances. Subject s22 made a large proportion of D1 choices, made fairly high-degree selections, made few close choices, and made some redundancy avoidance. See Table 6.3 for details.

Table 6.3: Summary performance results for Search condition subjects s5 and s22. Mean Δ-Opt was calculated as solution size - optimal solution size. % Opt Sol is the percent of solutions found that were optimal. Mean Selections is the mean number of (non-adjusted) selections used to find a solution. Mean D1 is the mean proportion of selections that were D1 vertices. Mean Distance is the mean path length between subsequent selections. Mean Degree is the mean Degree of selections normalized by the number of selections made. Mean RA is the mean proportion of selections that could be explained as Redundancy Avoidance selections.

Subject	**Mean Δ-Opt**	**% Opt Sol**	**Mean Selections**	**Mean D1**	**Mean Distance**	**Mean Degree**	**Mean RA**
s5	0	100.0%	116.28	0.2745	1.068	0.604	0.282
s22	1.32	32.0%	31.8	0.455	2.582	0.605	0.517

6.3.2 Evaluation of Models I and II

Model I was fit to subjects s17, s20, and s84 by running it with all possible combinations of parameter settings, and selecting the parameters which best matched the subjects' performance as follows. For each set of parameter settings, the model was run 100 times on each instance and the mean difference in size between the model's solution and the subject's solution was calculated. The total mean difference was calculated across all 25 instances. For all possible combinations of parameters, the 10-parameter sets with the lowest mean difference in size were saved. From this set, both the difference in solution size and number of selections needed by the model to find the solutions were compared, and a model was chosen from the 10 best fits based on both these measures. For the most part, the set of 10 best fits consisted of models which were very close in terms of mean solution size, differing by only a few selections over all 25 instances.

Model II was fit very similarly to Model I, except that two additional parameters were considered: persistence and undo-depth. These two parameters were adjusted until the model best fit the solution size measure. Then the 10 best fitting vertex choice parameter sets were determined and the best fit was determined on both solution size difference and the number of selections needed.

Finally, the model's solution to the large instances (instances 4, 9, 14, 19, 24) was compared to the subject's solution. Since the model's solution was based on 100 runs, the model's solutions were derived based on the likelihood that a vertex was included in the model's final solution and compared to the subject's solution. The aim of this analysis was to investigate how similar the models' solutions were to those found by the subject. This comparison included the size of the solution, the proportion of optimal solutions, the number of selections needed, and the number of Greedy and D1 choices in the final solution.

The Vertex Cover Model I

In terms of the mean size of solutions, Model I was able to fit subjects' performance very closely, with a difference of less than one vertex per instance overall. Model fit, as measured by the number of selections needed to find these solutions, was poorer and varied more greatly between subjects. The best fit in terms of solution size was to subject s17, a subject characterized by a large proportion of optimal solutions, very close to optimal solutions, and a number of selections which was generally very close to the size of the final solution. Evaluated in terms of selections, the model best fit subject s20, a subject characterized by few optimal solutions, solutions which were not close to optimal, and whose selections were exactly equal to the size of the solution on every instance. These results are summarized in Table 6.4. For each participant, the model's solutions to large instances (instances 4, 9, 14, 19, and 24) were compared to participant's solutions. The model was run 100 times on each instance, and the vertices that constituted the final solution on each of these runs was returned. The total number of times each vertex in the graph was included in a solution was tallied. These values were compared to the vertices included in the participant's solutions. A vertex was treated as a match if it was included in at least 60/100 of the runs.

Model fit to subject s17

The parameters which achieved the best fit for subject s17's solution size were: $S_{D1} = 0.9$, $S_{Gr} = 0.0$, $S_{Cl} = 0.1$, and $S_{RA} = 0.0$. Model I very closely matched

Table 6.4: Summary fit of Model I to subjects s17, s20 and s84. Solution Size Difference is the difference between Model I's solution size and the subject's, summed across all instances. Selection Difference is the difference between Model I's selections and the subject's, summed across all instances. Percent Optimal Difference is the difference between the percentage optimal solutions found by Model I, and that found by the subject, with positive values indicating the model found more optimal solutions than the subject.

Subject	**Solution Size Difference**	**Selection Difference**	**% Optimal Difference**
s17	11.21	360.55	+10.72
s20	16.95	16.95	+6.08
s84	15.66	1156.82	−1.28

s17's performance in terms of solution size, finding solutions of the same size (less than one vertex difference) on all but three instances. Of these instances, instance 11 and 12 were both Greedy Resistant, and instance 19 was large (39 vertices) with no D1 vertices, and not Greedy Resistant. Model solution size differed on these three instances by less than two vertices. These results are presented in Table 6.5. The subject found an optimal solution on 17/25 (or 68%) of all instances, whereas the model found optimal solutions 78.72% of the time. The instances on which the model found optimal solutions differed from the subject and there is no correlation between the proportion of optimal solutions found by the model on all runs on an instance, and whether or not s17 found an optimal solution on that instance (correlation = 0.018). In particular, the model found optimal with low likelihood on Greedy Resistant instances, whereas s17 found optimal solutions on three out of five of these Greedy Resistant instances. See Table 6.5 for details.

Model I was a poor fit to s17 in terms of the number of moves or selections needed to find a solution. Overall, the subject used more selections to find a solution, most notably on most large instances (9, 14, 19, and 24), and the Randomly Generated instances (20-24). These results are presented in Figures 6.1, and 6.2.

A refined analysis of solutions to the large instances indicates that the model finds solutions which are similar to the subjects' final solutions. An overview of a comparison between the vertices included in the subject's and model's solutions to the large instances is presented in Table 6.6.

On instance 4, subject s17 found an optimal solution, and made D1 strategy moves

Table 6.5: Subject s17 model fit. For each instance, the model's mean solution size, number of touches (selections), and percent optimal solutions are shown in comparison to the subject's actual values.

Instance	Model Size	s17 Size	Model Touches	s17 Touches	Model %-Opt	s17 Is Opt
0	6.01	6	6.19	7	99	YES
1	6.04	7	6.40	8	96	NO
2	9.19	9	9.65	9	83	YES
3	10.04	10	10.68	10	97	YES
4	17.00	17	17.70	18	100	YES
5	7.01	7	7.55	8	99	YES
6	9.10	9	9.18	9	90	YES
7	15.02	16	16.24	27	98	NO
8	16.03	17	17.69	21	97	NO
9	21.31	21	22.79	39	72	YES
10	6.42	6	7.58	7	58	YES
11	12.02	11	12.80	14	0	YES
12	11.22	10	12.66	11	39	YES
13	10.81	11	14.67	15	70	NO
14	21.92	22	24.08	58	9	NO
15	8.02	8	8.68	9	98	YES
16	5.00	5	5.34	6	100	YES
17	11.05	11	11.31	14	95	YES
18	13.51	13	14.07	26	51	YES
19	21.19	23	22.21	43	82	NO
20	7.07	8	7.35	17	93	NO
21	7.10	7	7.74	18	90	YES
22	13.09	13	13.91	33	91	YES
23	13.18	13	13.64	42	82	YES
24	18.28	19	19.72	64	79	NO

early in the problem-solving process. All D1 vertices were included by the model in the final solutions in 100 of 100 runs. The model's vertex choices match participant selections on 12 of 17 vertices in the final solutions (vertices included in > 60 of 100 runs). On this instance, the model also closely matches the subject's selections to find a solution, using 17.7 touches on average, where the subject used 19. The model, like the subject, included all D1 and the same two of three High Degree vertices in its final solutions. No vertex toggles were found in this instance's selections. See Figure A.51 for the subject's solution.

On instance 9, the subject found an optimal solution in 49 selections and made

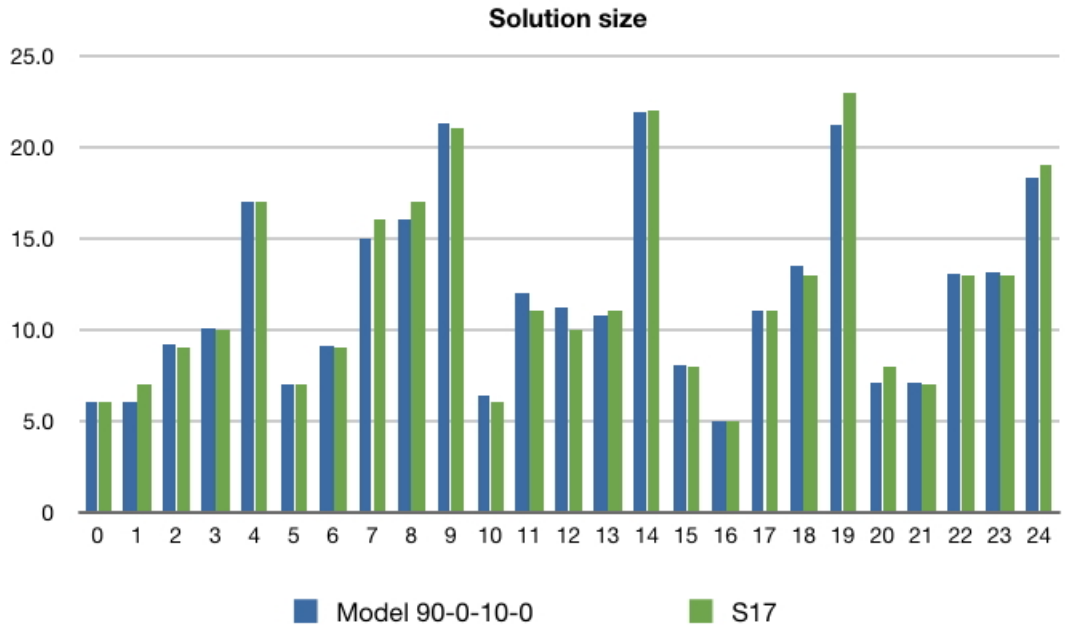


Figure 6.1: Comparison of subject s17 and Model I solution size.

high degree selections early during the process. The model's vertex choices that were included in the final solution matched participant selections on 18 of 21 vertices, and, like the participant, included a large number of high-degree vertices in the final solution. The model found a slightly less optimal solution, but to do so required many fewer selections (22.79). The model included the same three of four high-degree vertices in its final solution. This subject performed 12 vertex toggles, and therefore $49 - (12 * 2) = 25$ of the subject's selections are definitively selections. See Figure A.52 for the subject's solution.

Subject s17 found a sub-optimal but minimal solution to the Greedy Resistant instance 14 in 66 selections. The model's vertex choices matched participant selections on 15 of 22 vertices. On average the model was able to find slightly more optimal solutions in far fewer selections (24.08). Of the subject's selections, seven were vertex toggles, and therefore $66 - (7 * 2) = 42$ of the subject's selections are definitively selections. The model only included one of three high-degree vertices in its final solution, unlike the subject who included all three. See Figure A.53 for the subject's solution.

On instance 19, subject s17 found a sub-optimal but minimal solution, making few Greedy choices at the onset, and making 51 selections. The model's solutions

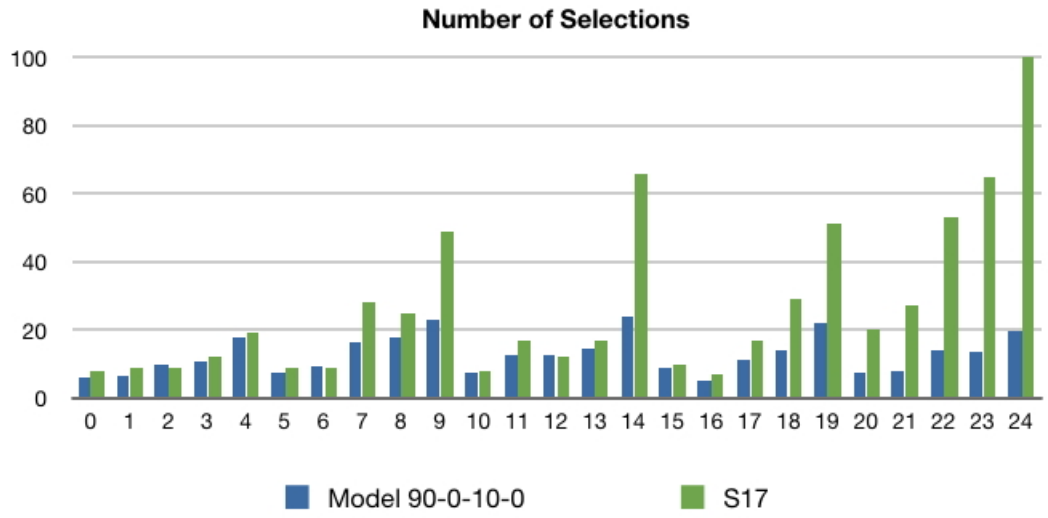


Figure 6.2: Comparison of subject s17 and Model I selections.

matched only 15 of 23 vertices in the final solution and made Greedy choices to start. On this instance, the model finds a better quality solution in far fewer moves (22.21) than the subject. Of the subject's selections, eight were vertex toggles, and therefore $51 - (8 * 2) = 35$ of the subject's selections are definitively selections. The model, like the subject, included all D1 and high-degree vertices in the final solutions. See Figure A.54 for the subject's solution.

Subject s17 found a sub-optimal but minimal solution to instance 24, taking many moves to settle upon a solution. On this instance, the subject made 101 selections before finalizing their solution. The model, in contrast, found a slightly less sub-optimal solution in only 19.72 moves. Of the subject's selections, seven were vertex toggles, and therefore $101 - (37 * 2) = 27$ of the subject's selections are definitively selections.The vertices included in the model's solutions matched the participants on 13 of 19 vertices, and included more high-degree vertices than the subject. See Figure A.55 for the subject's solution.

Model fit to subject s20

The model was originally fit to subject s20's data, with good success in terms of solution size; however, based on the observation that this subject's number of selections was exactly equal to the size of the final solution, the model was refit without the Optimization stage. This modification significantly reduced the difference of the model's solution size and number of touches needed to find a solution. The

Table 6.6: Summary comparison of Model's solutions to subject s17's solutions for large instances of each type.

Instance	s17 D1	s17 High Degree	Model D1	Model High Degree	Model /s17 Vertices
4	4/4	2/3	4/4	2/3	12/17
9	*na*	3/4	*na*	3/4	18/21
14	*na*	1/3	*na*	3/3	15/22
19	*na*	2/2	*na*	2/2	15/23
24	3/3	2/2	3/3	2/2	13/19

parameters which achieved the best fit with this modified model for subject s20's solution size were: $S_{D1} = 0.4$, $S_{Gr} = 0.3$, $S_{Cl} = 0.3$, and $S_{RA} = 0.0$. Model I closely matched s20's performance in terms of solution size, finding solutions of the same size (less than one vertex difference) on all but 10 instances. These 10 instances range in size from small to large, are dispersed across all instance types, and never differ by more than two vertices. These results are presented in Table 6.7. The subject found optimal solutions on only 5 of 25 instances (20%), whereas the model found optimal solutions 26.08 of instances. There is weak correlation between the proportion of optimal solutions found by the model on all runs on an instance, and whether or not s20 found an optimal solution on that instance (correlation = 0.4014). While the model found an optimal solution on instance 1 71% of the time, subject s20 did not find an optimal solution on this instance, and in fact their solution differed in size by two vertices. Similarly, although subject s20 found an optimal solution to instance 20, the model only found optimal solutions on this instance on 36 of 100 runs.

Model I very closely fit to s20 in terms of the number of moves or selections needed to find a solution. This fit precisely matches the fit in terms of solution size, as a direct result of the modified Model I, with no Optimization stage.

A refined analysis of solutions to the large instances indicates that the model finds solutions which are somewhat similar to the subjects' final solutions. This subject's selections, being exactly equal to the size of the solution, included no vertex toggles. An overview of a comparison between the vertices included in the subject's and model's solutions to the large instances is presented in Table 6.8.

On instance 4, subject s20 found an optimal solution and made D1 and Greedy strategy moves early in the problem-solving process. All D1 vertices were included by the model in the final solutions in 100 of 100 runs. The model's vertex choices match

Table 6.7: Subject s20 model fit. For each instance, the model's mean solution size, number of touches (selections), and percent optimal solutions are shown in comparison to the subject's actual values.

Instance	Model Size	s20 Size	Model Touches	s20 Touches	Model %-Opt	s20 Is Opt
0	6.29	6	6.29	6	81	YES
1	6.47	8	6.47	8	71	NO
2	10.51	9	10.51	9	26	YES
3	11.12	10	11.12	10	45	YES
4	18.06	17	18.06	17	34	YES
5	8.16	8	8.16	8	36	NO
6	10.09	10	10.09	10	40	NO
7	16.54	16	16.54	16	25	NO
8	18.19	17	18.19	17	11	NO
9	23.79	24	23.79	24	8	NO
10	7.47	7	7.47	7	3	NO
11	13.29	12	13.29	12	0	NO
12	12.84	14	12.84	14	1	NO
13	13.57	14	13.57	14	1	NO
14	24.87	25	24.87	25	0	NO
15	8.72	9	8.72	9	53	NO
16	5.45	6	5.45	6	61	NO
17	12.05	12	12.05	12	40	NO
18	15.03	14	15.03	14	4	NO
19	23.50	23	23.50	23	12	NO
20	8.20	7	8.20	7	36	YES
21	8.07	8	8.07	8	25	NO
22	14.63	15	14.63	15	16	NO
23	14.61	14	14.61	14	19	NO
24	20.89	22	20.89	22	4	NO

participant selections on 10 of 17 vertices in the final solutions (vertices included in > 60 of 100 runs). On this instance, the model also very closely matched the subject's selections to find a solution, using 6.29 touches on average, where the subject used 6. The model, like the subject, included all four D1 and all three high-degree vertices in its final solution. See Figure A.56 for the subject's solution.

On instance 9, the subject found a sub-optimal, non-minimal solution in 24 selections, and made high-degree selections early in the process. The model's vertex choices included in the final solution matched participant selections on only 12 of 24 vertices, and like the participant included a fairly large number of high-degree ver-

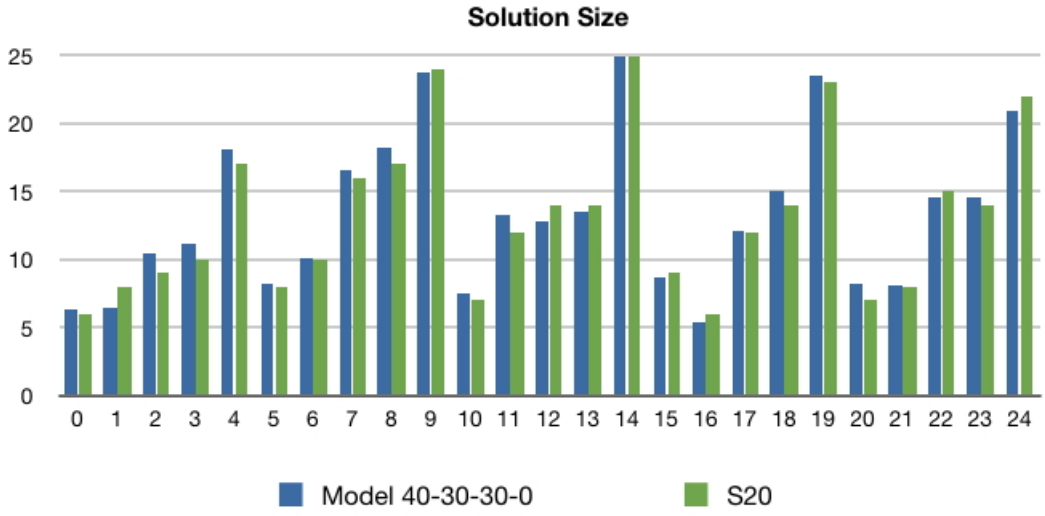


Figure 6.3: Comparison of subject s20 and modified Model I solution size.

tices in the final solution. The model included three to four high-degree vertices in the final solution, unlike the subject, who included all four. See Figure A.57 for the subject's solution.

Subject s20 found a sub-optimal and non-minimal solution to the Greedy Resistant instance 14, in 25 selections, made greedy choices early on in the process of finding a solution, and included all three highest degree vertices in the final solution. The model's vertex choices matched participant selections on 12 of 25 vertices, but like the subject included all three highest degree vertices in its final solution with high probability. Both the model and the subject included all three high-degree vertices in the final solution. See Figure A.58 for the subject's solution.

On instance 19, subject s20 found a sub-optimal and non-minimal solution, with Greedy choices at the onset and making 23 selections. The model's solutions matched only 17 of 23 vertices in the final solution. Both the model and the subject included both high-degree vertices in their final solution. See Figure A.59 for the subject's solution.

Subject s20 found a sub-optimal but minimal solution to instance 24, taking many moves to settle upon a solution. On this instance, the subject made 22 selections before finalizing their solution. The final solution includes all D1 vertices and does not include all high-degree vertices in their solution. The model, in contrast, found a more optimal solution in only 20.89 moves. The vertices included in the model's

Figure 6.4: Comparison of subject 20 and modified Model I selections.

solutions matched the participant's on 13 of 22 vertices and included more high-degree vertices than the subject. The model, like the subject, included all three D1 vertices in the final solution; however, the model included both high-degree vertices, where the subject included only one. See Figure A.60 for the subject's solution.

Table 6.8: Summary comparison of Model I's solutions to subject s20's solutions for large instances of each type.

Instance	**s20 D1**	**s20 High Degree**	**Model D1**	**Model High Degree**	**Model /s20 Vertices**
4	4/4	3/3	4/4	3/3	10/17
9	*na*	3/4	*na*	4/4	12/24
14	*na*	3/3	*na*	3/3	12/25
19	*na*	2/2	*na*	2/2	17/23
24	3/3	1/2	3/3	2/2	13/22

Model fit to subject s84

The strategy parameters which resulted in the best fit with the model for subject s84's solution size were: $S_{D1} = 0.1$, $S_{Gr} = 0.4$, $S_{Cl} = 0.5$, and $S_{RA} = 0.0$. Model I closely matched s84's performance in terms of solution size, finding solutions of the same size (less than one vertex difference) on all but five instances. Of these five instances (10, 11, 12, 22, 24), instances 10, 11, and 12 are Greedy Resistant, and

instances 22 and 24 are Randomly Generated. These results are presented in Table 6.5. The subject found optimal solutions on 12 of 25 instances (48%), where the model found optimal solutions on 46.72 of instances. There is a moderate correlation between the proportion of optimal solutions found by the model on all runs on an instance and whether or not s84 found an optimal solution on that instance (correlation = 0.5034). The model deviates most notably from the participant on the small Greedy Resistant instances (10, and 11) and instance 23 on which the model found optimal solutions with low probability. See Table 6.9 for details.

Table 6.9: Subject s84 model fit. For each instance, the model's mean solution size, number of touches (selections), and percent optimal solutions are shown in comparison to the subject's actual values.

Instance	**Model Size**	**s84 Size**	**Model Touches**	**s84 Touches**	**Model %-Opt**	**s84 Is Opt**
0	6.13	6	7.35	12	91	YES
1	6.17	6	6.75	8	85	YES
2	9.73	9	13.87	37	50	YES
3	10.65	10	14.15	19	61	YES
4	17.53	17	21.89	28	62	YES
5	7.62	8	10.38	46	54	NO
6	9.69	10	11.81	46	45	NO
7	15.45	15	21.09	49	64	YES
8	16.35	16	22.85	57	80	YES
9	22.09	22	29.65	98	31	NO
10	7.05	6	8.45	42	21	YES
11	12.07	11	15.79	75	13	YES
12	12.08	11	13.82	74	9	NO
13	12.29	12	17.11	85	35	NO
14	23.60	24	28.96	40	4	NO
15	8.15	8	10.41	34	88	YES
16	5.29	5	6.67	9	86	YES
17	11.51	12	14.25	23	55	NO
18	14.11	15	18.37	37	14	NO
19	22.32	23	29.38	59	24	NO
20	7.69	8	10.85	32	46	NO
21	7.85	8	11.31	40	47	NO
22	13.55	16	18.27	63	53	NO
23	13.87	13	17.31	27	22	YES
24	19.24	21	26.66	141	28	NO

Model I fits s84 very poorly in terms of the number of moves or selections needed

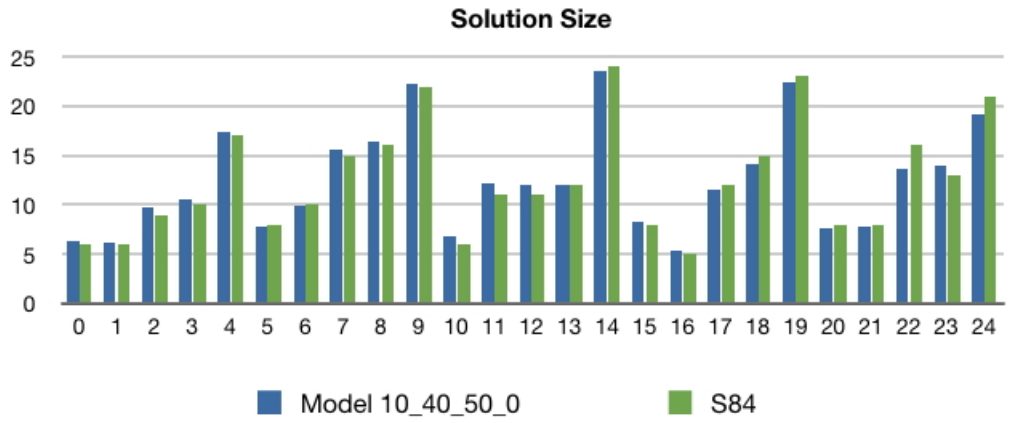


Figure 6.5: Comparison of subject s84 and Model I solution size.

to find a solution. While the participant makes a very large number of moves before settling upon a solution, the model uses only a few more moves than the size of the solution. This is a direct result of the way the model is implemented, as the only moves which will take place in excess of the solution size are those in the Optimization stage. See Figure 6.6 for details.

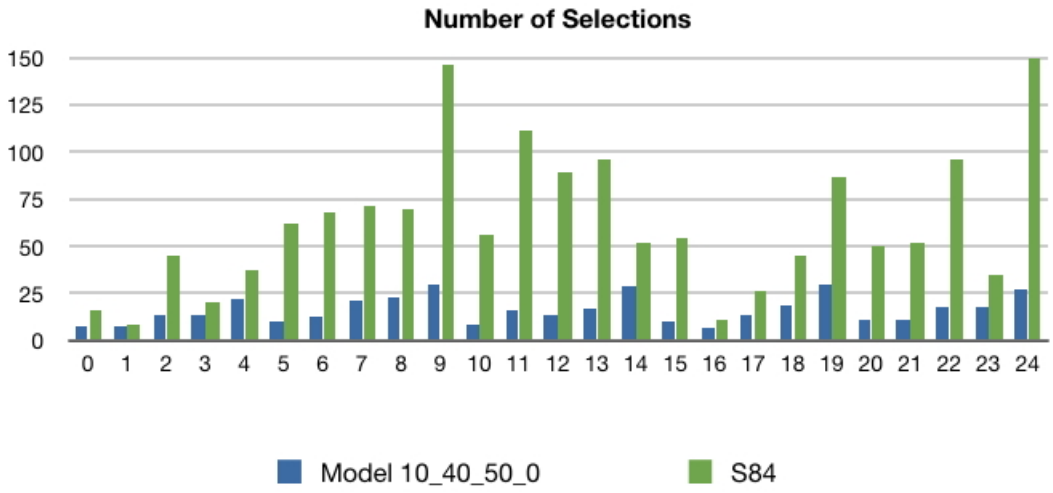


Figure 6.6: Comparison of subject s84 and Model I selections.

A refined analysis of solutions to the large instances indicates that the model finds solutions which are somewhat similar to the subjects' final solutions. An overview of

a comparison between the vertices included in the subject's and model's solutions to the large instances is presented in Table 6.10.

On instance 4, subject s84 found an optimal solution and made D1 early in the problem-solving process. The model's vertex choices match participant selections on 10 of 17 vertices in the final solutions. On this instance, the model used a fraction of the selections as subject s84 to find a solution, using only 7.35 touches in contrast to s84's 64. Of the subject's selections, nine were vertex toggles, and therefore $64 - (9 * 2) = 46$ of the subject's selections are definitively selections. The model, like the subject, included all D1 and high-degree vertices in its final solution. See Figure A.61 for the subject's solution.

On instance 9, the subject found a sub-optimal but minimal solution in 146 selections. The model's vertex choices included in the final solution matched participant selections on only 15 of 22 vertices and like the participant included a fairly large number of high-degree vertices in the final solution. The size of the model's solutions were a very close match (22.26 vertices), but unlike the subject, only required 29.6 selections on average to find its solutions. Of the subject's selections, 48 were vertex toggles, and therefore $146 - (48 * 2) = 50$ of the subject's selections are definitively selections.The model included all high-degree vertices in its solution, unlike the subject who included only three. See Figure A.62 for the subject's solution.

Subject s84 found a sub-optimal and minimal solution to the Greedy Resistant instance 14, in 52 selections. The model's vertex choices matched participant selections on 11 of 25 vertices, but like the subject included all three highest-degree vertices in its final solution with high probability. The model's solution's size was slightly smaller (23.53 vertices), and required only 28.96 selections, on average, to find. Of the subject's selections, 12 were vertex toggles, and therefore $52 - (12 * 2) = 38$ of the subject's selections are definitively selections. Both the model and subject included all high-degree vertices in their solutions. See Figure A.63 for the subject's solution.

On instance 19, subject s84 found a sub-optimal but minimal solution and made 87 selections. The model's solutions matched only 15 of 23 vertices in the final solution, requiring on average 29.38 selections. The model, like the subject, included all high-degree vertices in its final solutions. Of the subject's selections, seven were vertex toggles, and therefore $161 - (21 * 2) = 119$ of the subject's selections are definitively selections. This subject also included one full undo, in 43 steps. In essence, the subject added 43 vertices to their candidate cover, removed every one, then added and removed vertices, toggling vertices on and off during the process until a solution

was found. See Figure A.64 for the subject's solution.

Subject s84 found a sub-optimal but minimal solution to instance 24, in 161 selections. The model, in contrast, found a more optimal solution in only 26.66 moves. The vertices included in the model's solutions matched the participant's on 11 of 21 vertices. The model, like the subject, included all D1 and high-degree vertices in its solution. See Figure A.65 for the subject's solution.

Table 6.10: Summary comparison of Model I's solutions to subject s84's solutions for large instances of each type.

Instance	**s84 D1**	**s84 High Degree**	**Model D1**	**Model High Degree**	**Model /s84 Vertices**
4	4/4	3/3	4/4	3/3	10/17
9	*na*	3/4	*na*	4/4	15/22
14	*na*	3/3	*na*	3/3	11/25
19	*na*	2/2	*na*	2/2	15/23
24	3/3	2/2	3/3	2/2	11/21

The Vertex Cover Model II

In terms of the mean size of solutions, Model II was able to fit subjects' performance very closely, with a difference of less than one vertex per instance overall. Model fit as measured by the number of selections needed to find these solutions, however, was poorer, and varied more greatly between subjects. The best fit in terms of solution size was to subject s5, a subject characterized by all optimal solutions and a great number of selections to find a final solution. Evaluated in terms of selections, the model best fit subject s22, a subject characterized by few optimal solutions, solutions which were not close to optimal, and whose number of selections were close to the size of the solution on most instances. These results are summarized in Table 6.11. For each participant, the model's solutions to large instances (instances 4, 9, 14, 19, and 24) were compared to participant's solutions. The model was run 100 times on each instance, and the vertices which constituted the final solution on each of these runs was returned. The total number of times each vertex in the graph was included in a solution was tallied. These values were compared to the vertices included in the participant's solutions. A vertex was treated as a match if it was included in at least 60/100 of the runs.

Table 6.11: Summary fit of Model II to subjects s5 and s22. Solution Size Difference is the difference between Model II's solution size and the subject's, summed across all instances. Selection Difference is the difference between Model II's selections and the subject's, summed across all instances. Percent Optimal Difference is the difference between the percentage optimal solutions found by Model II, and that found by the subject, with positive values indicating the model found more optimal solutions than the subject.

Subject	**Solution Size Difference**	**Selection Difference**	**% Optimal Difference**
s5	0.25	1905.41	0%
s22	21.76	47.38	+29.4%

Model fit to subject s5 The strategy parameters which resulted in the best fit with the model for subject s84's solution size were: $S_{D1} = 0.5$, $S_{Gr} = 0.0$, $S_{Cl} = 0.4$, and $S_{RA} = 0.1$, with a Persistence of 50, and an Undo Depth of 100%. Model II nearly precisely matched s5's performance in terms of solution size, finding solutions of the same size on all but four instances. Of these four instances (10, 11, 12, 13), all are Greedy Resistant, and the model's solution size was still less than one out of 10 of a vertex larger (taken as a mean across 100 runs). These results are presented in Table 6.12. The subject found optimal solutions on all instances (100%) and the model found optimal solutions 99.28% of instances. Since there is no variation in the subject's data, it is not possible to calculate correlation between the proportion of optimal solutions found by the model on all runs on an instance, and whether or not s5 found an optimal solution on that instance. The model deviates most notably from the participant on the Greedy Resistant instances, as seen in Figure 6.7.

Overall, model II fits subject s5 very poorly in terms of the number of moves or selections needed to find a solution. On most instances, subject s5 makes a very large number of moves before settling upon a solution, whereas the model, which uses many more moves than the size of the solution, finds a solution in fewer selections. This pattern, however, changes on the Greedy Resistant instances and the large Randomly Generated instances, where subject s5 requires far fewer selections to find an optimal solution than the model. See Figure 6.8 for details.

A refined analysis of solutions to the large instances indicates that the model frequently finds solutions which are very similar to the subjects' final solutions. An overview of a comparison between the vertices included in the subject's and model's

Table 6.12: Subject s5 model fit. For each instance, the model's mean solution size, number of touches (selections), and percent optimal solutions are shown in comparison to the subject's actual values.

Instance	Model Size	s5 Size	Model Touches	s5 Touches	Model %-Opt	s5 Is Opt
0	6	6	21.96	26	100	YES
1	6	6	26.46	47	100	YES
2	9.04	9	71.18	41	97	YES
3	10.02	10	67.08	67	98	YES
4	17	17	77.16	119	100	YES
5	7.01	7	42.33	19	99	YES
6	9.01	9	59.49	48	98	YES
7	15.02	15	73.88	68	99	YES
8	16.01	16	78.95	133	99	YES
9	21.08	21	225.82	263	94	YES
10	6.22	6	64.90	23	76	YES
11	11.36	11	164.18	47	73	YES
12	10.3	10	96.28	55	85	YES
13	10.6	10	124.00	64	85	YES
14	20.469999	20	238.31	62	79	YES
15	8	8	25.82	59	100	YES
16	5.06	5	29.54	7	97	YES
17	11.02	11	68.32	33	98	YES
18	13.23	13	171.65	253	81	YES
19	21.24	21	264.72	180	88	YES
20	7	7	45.38	49	100	YES
21	7.03	7	58.79	146	98	YES
22	13.04	13	110.42	281	95	YES
23	13.11	13	148.15	239	86	YES
24	18.120001	18	205.44	71	93	YES

solutions to the large instances is presented in Table 6.13.

On instance 4, subject s5 found an optimal solution. The model's vertex choices match participant selections on 16 of 17 vertices in the final solutions (vertices included in 60 of 100 runs). On this instance, the model used fewer selections than subject s5 to find a solution, using only 52.58 (mean) touches in contrast to s5's 135. Of the subject's selections, seven were vertex toggles, and therefore $135-(7*2) = 121$ of the subject's selections are definitively selections. This subject performed two long undos (one of length 31, one of length 26). The model, like the subject, included all D1 and high-degree vertices in its final solution. See Figure A.66 for the subject's

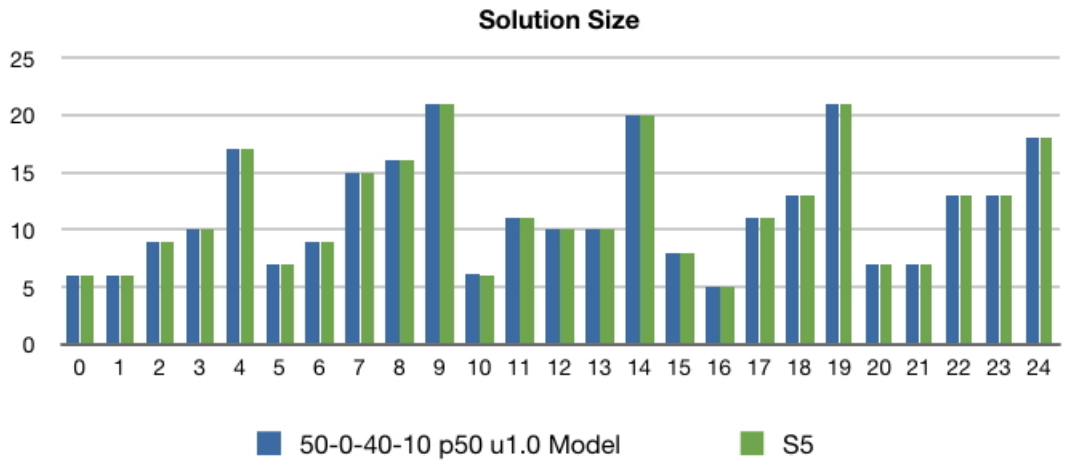


Figure 6.7: Comparison of subject s5 and Model II solution size.

solution.

On instance 9, the subject found an optimal solution in 355 selections. The model's vertex choices included in the final solution matched participant selections on 16 of 21 vertices and like the participant included a large number of high-degree vertices in the final solution. While the model did not use as many selections as the subject, it took a great deal of search, 163.22 mean selections, to find the solution. Of the subject's selections, 49 were vertex toggles, and therefore $355 - (49 * 2) = 257$ of the subject's selections are definitively selections. The subject also included two full undos of lengths 35 and 27 and one short backtracking sequence of length 7. Both the model and the subject included all four high-degree vertices in their final solutions. See Figure A.67 for the subject's solution.

Subject s5 found an optimal solution to the Greedy Resistant instance 14, in 82 selections. The model's vertex choices matched participant selections on 20 of 20 vertices. Both the model and subject included no high-degree vertices in their final solutions. The model required many more selections, 391.35 selections on average, to find a solution. Of the subject's selections, 19 were vertex toggles, and therefore $82 - (19 * 2) = 64$ of the subject's selections are definitively selections. See Figure A.68 for the subject's solution.

On instance 19, subject s5 found an optimal solution making 231 selections. The model's solutions matched 15 of 21 vertices in the final solution, requiring on average 191.60 selections. While the subject included only one high degree vertex in its

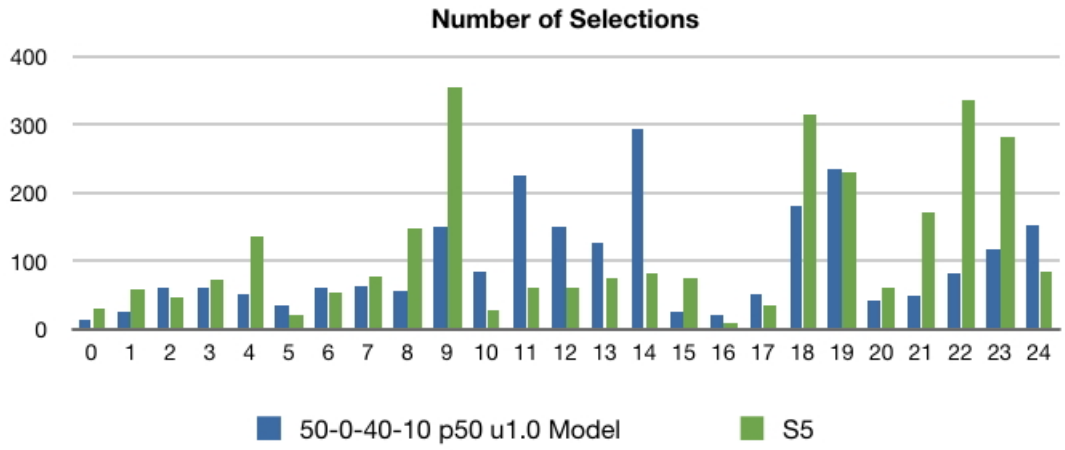


Figure 6.8: Comparison of subject s5 and Model II selections.

solution the model included both. Of the subject's selections, 56 were vertex toggles, and therefore $231-(56*2) = 119$ of the subject's selections are definitively selections, possibly meaning the subject used few selections to find its solution than the model did on average. See Figure A.69 for the subject's solution.

Subject s5 found an optimal solution to instance 24 in 84 selections. The model required more selections, 131.42, to find that solution. These results are presented in Table 6.9. The vertices included in the model's solutions matched the participant's on 12 of 18 vertices. Of the subject's selections, 11 were vertex toggles, and therefore $84-(11*2) = 62$ of the subject's selections are definitively selections. The model, like the subject, included all D1 and high-degree vertices in its final solution. See Figure A.70 for the subject's solution.

Table 6.13: Summary comparison of Model II's solutions to subject s5's solutions for large instances of each type.

Instance	**s5 D1**	**s5 High Degree**	**Model D1**	**Model High Degree**	**Model /s5 Vertices**
4	4/4	3/3	4/4	3/3	16/17
9	*na*	4/4	*na*	4/4	16/21
14	*na*	0/3	*na*	0/3	20/20
19	*na*	1/2	*na*	2/2	15/21
24	3/3	2/2	3/3	2/2	12/18

Model fit to subject s22

The strategy parameters which resulted in the best fit with the model for subject s22's solution size were: $S_{D1} = 0.5$, $S_{Gr} = 0.3$, $S_{Cl} = 0.1$, and $S_{RA} = 0.1$, with a Persistence of 2, and an Undo Depth of 20%. Model II poorly matched s22's performance in terms of solution size, finding solutions of the same size on all but nine instances (6,7,9,10,12,19,22,23,24) (< 1 vertex different). Of these nine instances, only two differed by more than two vertices, instances 19 and 24, both large instances. These results are presented in Figure 6.9. The subject found optimal solutions on 8/25 instances (32%), and the model found optimal solutions 61.40% of instances. There is little correlation between the proportion of optimal solutions found by the model on all runs on an instance and whether or not s22 found an optimal solution on that instance (0.2855). See Table 6.14 for details.

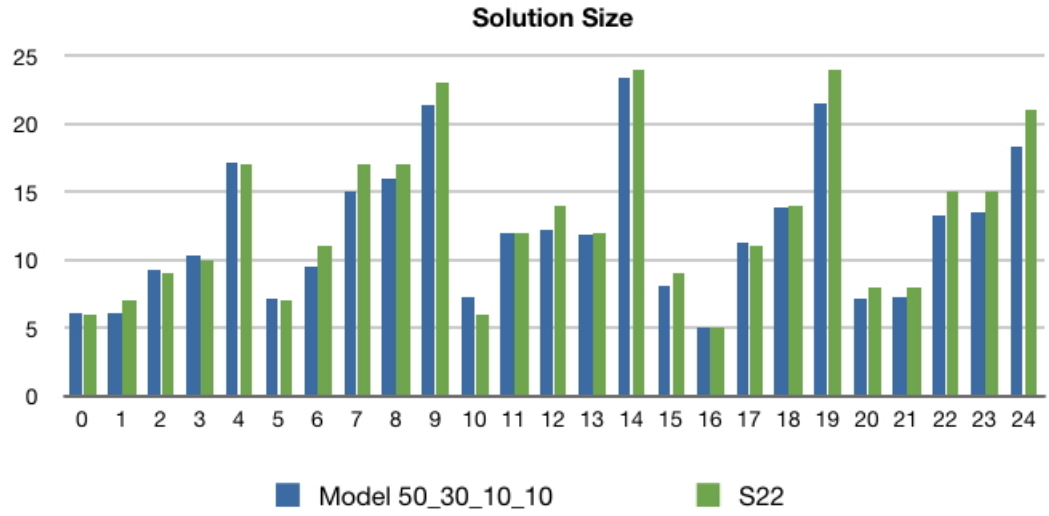


Figure 6.9: Comparison of subject s22 and Model II solution size.

Overall, model II fits subject s22 very well in terms of the number of moves or selections needed to find a solution. On most instances, subject s22 makes very few moves before settling upon a solution, which the model matches very closely on most instances. The model deviates from the participant's selections most notably on the large instances, instances 4, 9, 14, 19 and 24. See Figure 6.10 for details.

A refined analysis of solutions to the large instances indicates that the model frequently finds solutions which are very similar to the subjects' final solutions. An overview of a comparison between the vertices included in the subject's and model's

Table 6.14: Subject s22 model fit. For each instance, the model's mean solution size, number of touches (selections), and percent optimal solutions are shown in comparison to the subject's actual values.

Instance	Model Size	s22 Size	Model Touches	s22 Touches	Model %-Opt	s22 Is Opt
0	6.04	6	6.42	6	96	YES
1	6.13	7	6.97	13	87	NO
2	9.31	9	11.91	9	70	YES
3	10.32	10	12.06	10	76	YES
4	17.12	17	19.70	21	88	YES
5	7.18	7	8.28	7	82	YES
6	9.47	11	11.09	11	53	NO
7	15.11	17	17.23	17	89	NO
8	16.08	17	18.74	21	94	NO
9	21.44	23	27.62	25	64	NO
10	7.06	6	9.02	6	24	YES
11	12.03	12	15.83	14	0	NO
12	12.18	14	16.08	16	6	NO
13	11.57	12	17.71	24	48	NO
14	23.47	24	33.39	30	3	NO
15	8.02	9	8.54	9	98	NO
16	5.11	5	5.67	5	95	YES
17	11.40	11	13.36	11	60	YES
18	13.85	14	18.09	16	24	NO
19	21.64	24	27.64	26	53	NO
20	7.21	8	8.37	10	79	NO
21	7.37	8	9.07	8	66	NO
22	13.27	15	15.43	15	75	NO
23	13.72	15	17.60	15	32	NO
24	18.34	21	23.20	25	73	NO

solutions to the large instances is presented in Table 6.15.

On instance 4, subject s22 found an optimal solution. The model's mean solution size is very close to optimal (17.12) and the model's vertex choices match participant selections on 15 of 17 vertices in the final solutions (vertices included in 60 of 100 runs). On this instance, the model used marginally more selections than subject s22 to find a solution, using 19.70 (mean) touches in contrast to s22's 21. Of the subject's selections, two were vertex toggles, and therefore $21 - (2 * 2) = 17$ of the subject's selections are definitively selections. The model and subject both included all D1 and high-degree vertices in their final solutions. See Figure A.71 for the subject's solution.

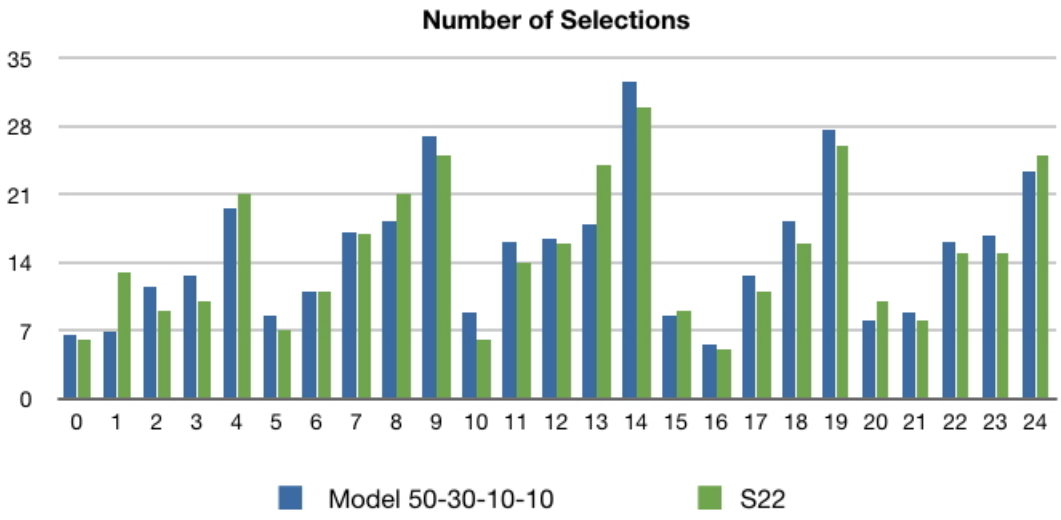


Figure 6.10: Comparison of subject s22 and Model II selections.

On instance 9, the subject found a sub-optimal and minimal solution in 25 selections. The model's vertex choices included in the final solution matched participant selections on 16 of 21 vertices. The model used slightly more selections than the subject, 27.62 mean selections, to find the solution. Of the subject's selections, zero were vertex toggles. The model, like the subject, included all four high-degree vertices in its final solution. See Figure A.72 for the subject's solution.

Subject s22 found a sub-optimal and minimal solution to the Greedy Resistant instance 14 in 25 selections. The model's vertex choices only matched participant selections on 6 of 24 vertices, and required slightly more selections, 33.39 selections, on average, to find a solution. Of the subject's selections, one was a vertex toggle, and therefore $25 - (1 * 2) = 21$ of the subject's selections are definitively selections, implying that this subject found a solution in significantly fewer selections than the model. While the subject only included two of three high-degree vertices in their final solution, the model included all three. See Figure A.73 for the subject's solution.

On instance 19, subject s22 found a sub-optimal and minimal solution in 26 selections. The model's solutions matched 9 of 24 vertices in the final solution, requiring on average 27.64 selections. Of the subject's selections, zero were vertex toggles. The model's solution was significantly smaller than the subject's, with a mean size of 21.64. The model again deviated from the subject in the vertices included in its solution, with both high-degree vertices in its final solution. See Figure A.74 for the

subject's solution.

Subject s22 found a sub-optimal and minimal solution to instance 24 in 25 selections. The model required fewer selections, 23.20, to find a better quality solution. Of the subject's selections, one was a vertex toggle, and therefore $25 - (1 * 2) = 21$ of the subject's selections are definitively selections, and the model may have required marginally more selections to find a solution. The vertices included in the model's solutions matched the participant's on 11 of 21 vertices. Both the model and subject included all three D1 vertices in their final solutions; however, the model included both high-degree vertices where the subject included only one. See Figure A.75 for the subject's solution.

Table 6.15: Summary comparison of Model II's solutions to subject s22's solutions for large instances of each type.

Instance	**s22 D1**	**s22 High Degree**	**Model D1**	**Model High Degree**	**Model /s22 Vertices**
4	4/4	3/3	4/4	3/3	15/17
9	*na*	4/4	*na*	4/4	16/21
14	*na*	2/3	*na*	3/3	6/25
19	*na*	1/2	*na*	2/2	9/24
24	3/3	2/2	3/3	1/2	11/21

6.4 Discussion

This chapter investigated the potential of two probabilistic models (Model I and Model II) to account for human performance on the Optimization and Search versions (respectively) of the hard computational problem, the Vertex Cover problem. Both models were based on simple, local strategies (both correct and heuristic), applied probabilistically. Model II, using Model I as a core, had two additional parameters to emulate the persistence and backtracking frequently observed when subjects were tasked with the Search version of the problem.

6.4.1 Solution Size

Based on the model's solution size, as is typical in attempts to model human performance on hard computational problems, both models did a good job of fitting human

performance on their associated version of Vertex Cover, supporting the hypothesis that performance on these two versions of the Vertex Cover problem can be modelled with simple local strategies.

The Vertex Cover Model I

In terms of solution size, Model I fit all three subjects well or very well, deviating from subjects' solution size by less than one vertex per instance, on average. This indicates that these local strategies may be representative of the strategies used by subjects. There was, however, some deviation in fit on instance type. The model was not able to account for s20's performance on those instances with no D1 vertices at the onset which were not greedy resistant, and similarly was not sufficient to capture s84's performance on Greedy Resistant instance. The model also deviated from the subjects' performance when broken down by instance size, failing to fit s17's performance on large instances. So, while the model appears to capture the subjects' performance by this measure, it is clear that it fails to capture some aspect of their performance which is dependent on instance size or type. Recall that the model works with statically set probabilities of selecting one of four strategies. One possible explanation for this difference in performance, based on the assumption that the model's strategies are representative of what subjects are doing, is that subjects are able to shift how likely they are to apply strategies. Consider s84, for instance, who was able to find closer-to-optimal solutions on Greedy Resistant instance than the model, which could be explained if this subject is able to reject (at least to some extent) the greedy strategy on these Greedy Resistant instance, whereas the model was stuck making greedy choices with its predetermined probability.

An alternate explanation of the model's good fit (to subjects s17 and 84), however, is that these subjects were highly likely to find minimal solutions. Since this problem as given is ill-defined, it is possible that participants, being unable to solve the problem as given, reformulate the problem into another problem for which minimal vertex covers were a suitable solution. The model, by design, always found minimal covers, and therefore the solutions ought to match. This explanation does not make any assumptions about the kinds of strategies subjects may be using to solve the given task, nor even the problem being solved, and therefore the model fit does not necessarily mean that the model's strategies are representative of those used by the subjects, only that the solutions were typically minimal. For instance, minimal covers

could be found solely by making random vertex selections and then applying the local optimization used in the model. This alternate explanation does not directly apply to subject s20 and the model, as this subject did not find minimal covers on most instances. The model was modified to better fit this subject, based on this observation, and lacked the optimization stage. As such, both the model and the subject found vertex covers, the model because of its very design, and the subject because the interface required a valid cover to proceed to the next instance. Therefore, it is hardly surprising that the model and subject match on this measure. However, this does not explain the closeness of fit, as there are many possible covers; therefore this alternate explanation is a less likely explanation of the model's fit to subject s20.

The Vertex Cover Model II

In terms of solution size, Model II fit both subjects well or very well, deviating from subjects' solution size by less than one vertex per instance, on average. More notable was the close fit of Model II to subject s5, which deviated by less than 0.25 vertices *total* summed across all instances. There was little difference in goodness of fit between instance type; however, the model fit s22's performance more poorly on large instances with a difference of more than one vertex on all large instances except the greedy resistant instance 14. The model appears to capture the subjects' performance by this measure, with the noted exception of out-performing subject s22 on four out of five of the large instances. Like Model I, this goodness of fit could be a result of the model being representative of how subjects solve instances of this problem. Unlike Model I, the model also appears to better capture the subjects' performance as influenced by instance size and type. This good fit could be interpreted as being because the strategies, persistence, and undoing are representative of how subjects solved instances of this problem.

However, it is also possible that the goodness of fit is a result of the kinds of solutions found, independent of the strategies used to achieve them. The reason for this stems from the well-defined goal given in the Search version of the Vertex Cover problem. The model by design attempts to find a solution which matches the given goal, making multiple attempts until the goal is met, or the model gives up. As a result, the model, like subject s5, finds optimal solutions very frequently. This explains the good fit between subject s5 and Model II, however, is not sufficient to explain the fit of this model to subject s22, who failed to find optimal solutions

with high frequency. The goodness of fit of Model II to subject s22 is probably better explained by the observation that this subject frequently finds minimal solutions, and the model, whether it finds optimal or gives up, will likewise always find a minimal solution.

Limitations

Both these models fit the participants' data perhaps surprisingly well in terms of solution size. However, it is very important to note that, unlike other hard computational problems that have been the subject of modelling like E-TSP, there is likely very little difference possible in solution size for this problem. In the Optimization condition, earlier results support the hypothesis that participants may be trying to find *minimal* vertex covers as a result of the goal modification that takes place. While there are possibly many different minimal covers for a given instance, for the size of instance used in this work, it is unlikely that these covers will deviate in size by more than three or four vertices. In this light, the close fit is perhaps less surprising. This also true for the fit of Model II. While there cannot be a range of values that could be interpreted as correct, and only one solution size that might match the goal, the model still works by finding minimal covers, which are going to constrained in terms of possible sizes. The fit of the model to subject s5 is undeniably close. However the fit of Model II to subject s22 is vulnerable to the same issue as that of Model I: the model, restricted to always finding minimal covers, will always be relatively close to optimal. This limitation does not impact the percent optimal solutions, however. Model I fits all three subjects well in terms of the percent optimal solutions found. This measure further supports the finding that Model II was able to very closely fit subject s5 in terms of solution quality.

6.4.2 Solution Properties

Based on the analysis of the properties of the solutions found by the model on large instances, the model does a good job of fitting subjects' solutions in both groups, again supporting the hypothesis that performance on these two versions of the Vertex Cover problem can be explained by the use of simple local strategies.

The Vertex Cover Model I

For all subjects, the model perfectly fit the number of D1 vertices that were included in the final solution. Fit was a bit poorer, but still overall good, in terms of the other two main measures: high-degree vertices and the match between the models' vertices and the subjects. This fit further supports the possibility that the model captures how subjects are finding their solutions, as in general the model's solutions are similar to all three subjects' solutions to large instances.

It is also possible that there is an alternate reason why the model and subjects' solutions share these properties. However, it is important to note that the instances were designed to make the likelihood very low of selecting these vertices as members of the cover by chance. There were only four and three D1 vertices in instances 4 and 24 respectively. Taken either as a proportion of total vertices, or a proportion of vertices in the solution, it is highly unlikely that all D1 vertices were included by chance by the model and subject. It is possible, however, that they were included in the final solution due to other strategies. The perfect match between model and subjects, given that the best fitting model for each subject included a D1 parameter value that was greater than one on this measure, is strong support for the hypothesis that subjects indeed intentionally choose D1 vertices when finding their solutions and that this strategy is included in the participants' toolbox.

Similarly, few high-degree vertices were included in each instance and slightly poorer but still good fit in terms of high-degree vertices is likely not due to chance. While this is true, there are still various other reasons these vertices could have been included in the solution. They could have been chosen by the model by random, by the Cl strategy, the D1 strategy (even if they were not D1 at the onset, they could have been at an intermediate state), or by the RA strategy. Similarly they could have been selected by those or other strategies by the subjects. It is also possible that the high degree and therefore relatively large number of edges connected to the vertex somehow drew the subjects' attention to these vertices more than other vertices, perhaps due to the Gestalt principle of Continuation. Therefore, while an alternate reason might explain their inclusion, this result supports the hypothesis that like the model, subjects purposely include high-degree vertices in their solutions.

The Vertex Cover Model II

The model very closely fit subject s5 and fit subject s22 moderately well, on all identified measures of solution property of the large instances. In particular, the model perfectly fit the number of D1 vertices included in the final solution of both subjects. The model's fit to subject s5 on both high degree and matching vertices was very good, which is likely related to the fact that the subject and model both found optimal solutions with very high likelihood and there are a limited number of optimal solutions for each instance. On instance 14, for instance, there is only one optimal solution and this explains the perfect match on all measures of solution property.

This fit supports the hypothesis that the model reflects strategies that subjects might be using to find solutions. However, it is also possible that the good fit is solely a result of the model and subjects finding solutions with similar qualities (size or optimality, for instance), independent of the types of selections or strategies that could lead to these solutions. It is therefore prudent to recognize that while the model may represent how subjects find solutions to this problem, there are other, equally viable, explanations as well.

6.4.3 Strategies

The models attempt to capture human performance on the Vertex Cover problem by using four local strategies, and were fit to each subject by identifying the set of parameter values associated with the best fit based on solution size. These parameters represented the probability that a selection would be made by each of four strategies to find a solution. If these strategies are representative of those used by subjects, then one would hope to find a correlation between the parameter values associated with the best fit, and the subjects' likelihood of applying those strategies. The actual strategies used by participants are unknown; however, they can be inferred from the qualities of vertices included in the final solution, and the qualities of vertices selected during the problem-solving process. This inference is affected by differences between how the model implements the strategy selection process, and how it might be implemented by the subjects. All subjects were naive to the given problem, and therefore if strategy adoption occurred, it would not be observed until after the subject interacted with one or more instance to which the strategy was applicable. One possible argument against this assumption is that subjects might think about an instance and come up with strategies which could be used starting

with the very first move. However, the speed with which vertices were selected is not in keeping with subjects thinking deeply about the problem and reasoning about strategies before making moves. Because of the difference between the model and subject strategy acquisition, it would not be surprising if the model, with its static allocation of strategy parameters, did not precisely match measures of the subjects' application of these strategies, even for strategies used by subjects. The subjects' measures on those strategies would be diluted or reduced, overall, by those moves or selections made before a strategy were recognized and acquired. It would expected, based on this observation, that if the model's strategies were representative of those used by a subject, then the model's parameter values should exceed the likelihood of subjects using that strategy. Interaction with instances resistant to a particular strategy could also alter the likelihood with which a learned strategy were applied on later strategies, a variability in strategy application which the model would not be able to capture. Basically the weakness here is comparing the overall likelihood of a strategy being applied by the model, given a static likelihood, and the overall likelihood of it being applied by the subject, given a variable likelihood (either increasing, decreasing, or variable in some other way). Therefore, even if the model's values are representative, in general, of the subject's likelihood of applying those strategies, there will be differences between the observed frequencies of applying the strategies, instance to instance, type by type, and also over time. So, while in general, the model's parameters do not fit the inferred strategies used by participants, this does not discount the possibility that the model's strategies are representative of those used by subjects.

The Vertex Cover Model I

There is some support for the hypothesis that the model captures the subjects' use of the D1 strategy. While the model's parameter values do not precisely match the subjects' frequency of choosing D1 vertices, the model, like all three subjects, included all D1 vertices in their final solutions to the large instances. On those instances for which optimal solutions were found, either by the model or subjects, these D1 vertices had to be included, and therefore the match is not necessarily an indication of the subject and model matching on this strategy. However, none of the three subjects found optimal solutions on instance 24, and therefore this match between the model and subjects on instance 24 supports the theory that participants do indeed learn and

apply the D1 strategy. The overall variation between the model's parameter value and subjects' likelihood of making D1 choices may be in part due to the subjects' ability to vary their strategy application from instance to instance either through the adoption or rejection of the strategy, which the model was not designed to handle.

The match between the model and participants' GR strategy does not strongly support the hypothesis that the model is representative of the subjects' use of the GR strategy. Overall, across all instances, there is little relationship between the parameter values and the individual subjects' likelihood of making high degree choices. Both the model and subjects, however, were highly likely to include the highest degree vertices in the solutions to large instances. This heuristic strategy is particularly sensitive to learning, as instances were developed and included were explicitly resistant to it, which may have impacted the subjects' likelihood of applying the strategy on subsequent instances, thus skewing the observed frequency of application of this strategy.

The match between the model and participants' CL strategies, supports, to a limited extent, the hypothesis that the model captures the subjects' use of this strategy. The CL parameter for all three participants is greater than zero, and all three participants made relatively close subsequent selections, hovering near two; however, the variation of model parameter and subjects' measures does not correlate.

A complete lack of match between the model and participants' application of the RA strategy does not support the hypothesis that the model captures the subjects' use of this strategy. The RA parameter for all three participants is equal to zero; however, the subjects' mean number of RA moves is greater than zero varies between participants. This is not strong support for model accurately reflecting the kinds of strategies subjects are using.

The model does not, based on a comparison between parameter values and the individual subjects' measures, clearly support the theory that participants are indeed using all four identified strategies. There are many possible explanations for this. The subjects' ability to vary how they select which strategy they use may explain this difference. It is also possible that participants do not use any of the identified strategies; however, this is not in keeping with participants' verbal description of using the D1 and GR strategy, nor with the close match between the number of D1 and high-degree vertices the model and subjects included in the solutions to the large instances. Finally, the model being able to closely match the subjects' performance on other measures using these strategies at least supports the theory that these strategies

are sufficient to explain human performance on this problem, even if the specific way the model is implemented may not be an accurate reflection of how subjects solve this problem, step by step.

The Vertex Cover Model II

There is some support for the hypothesis that this model captures the subjects' use of the D1 strategy. The model, like all subjects, included all D1 vertices in their final solutions to the large instances. On those instances for which optimal solutions were found, either by the model or subjects, these D1 vertices had to be included, and therefore the match is not necessarily an indication of the subject and model matching on this strategy. However, subject s22 did not find an optimal solution on instance 24, and therefore this match between the model and this subject on instance 24 supports the theory that participants do indeed learn and apply the D1 strategy. Differences between the model parameter values and subjects' likelihood of choosing D1 vertices can be explained by the subjects' ability to vary their strategy application from instance to instance, either through the adoption or rejection of the strategy, which the model was not designed to handle.

While the match between the model's parameter values and participants' frequency of selecting high-degree vertices overall does not strongly support the hypothesis that the model is representative of the subjects' use of the GR strategy, there is a good match between the model and subjects on the inclusion of high-degree vertices on large instances. This heuristic strategy is particularly sensitive to learning, as instances were developed and included to be explicitly resistant to it, which may have impacted the subjects' likelihood of applying the strategy on subsequent instances, thus skewing the observed frequency of application of this strategy. In this condition, subjects were able to compare the size of their solution to the size of the goal, and therefore had more indication of when a strategy was not successful. This could have manifested in an increased variability of strategy application.

There is good support for the hypothesis that this model captures the subjects' use of the CL strategy, as the model's parameter value varies in a similar way as the subjects' mean normalized difference between subsequent selections.

Finally, there is little support for the theory that the model's use of the RA strategy is representative of subjects' use of the same strategy.

The model, based on a comparison between parameter values and the individual

subjects' measures, supports the theory that participants are using at least three of the four identified strategies. In particular, there is support for the theory that subjects are making local choices, independent of the other measures, as indicated by the good match between the model and subjects' measures for the CL strategy. While the actual parameter values and subjects' frequency do not match for either the D1 or GR strategies, the more refined analysis of solutions to large instances suggests that the model captures the kinds of solutions found based on these measures.

6.4.4 Number of Selections

The Vertex Cover Model I

At first glance, the model does a poor job of capturing subjects' performance in terms of the number of selections needed to find a solution. With the exception of subject s20, the model found solutions in far fewer moves than the subjects, especially on large instances. However, further analysis of the selections made by participants on the large instances provides an explanation for this difference at least in part. It appears as if subjects' selections are inflated by their use of the interface to judge the impact of selections by toggling vertices on and off. If this indeed the case, then the model may in fact come close to matching the number of actual moves used by participants to find a solution.

The observed Toggling behaviour is likely not the only way that participants were using the interactive interface to *explore* the instances of this problem. Many other selections or sets of selections made by participants could equally be exploratory moves that the model was not designed to capture. With this in mind, it would be expected that the subjects' selections would exceed the model's, as the model is simply adding vertices to the candidate cover, whereas there are many possible reasons that motivate subjects to select a vertex. Notably, the number of selections used to find a solution is not frequently reported, as most other attempts to model human performance on hard computational problems have focused on other measures (solution quality, solution size, and time to find a solution). Perhaps as a result, this *exploratory* behaviour has not been reported in other studies of human performance on hard computational problems, and it is interesting to wonder if it is behaviour that is specific to the problem, the interface, or the some other factor.

The Vertex Cover Model II

This model does a poor job of capturing subjects' performance as measured by the number of selections needed to find a solution, in particular for subject s5. To some extent, this fit can be improved if we exclude from subjects' selections those which appear to be toggling or exploratory behaviour. But this fails to explain why this model required significantly more selections to find solutions of quality matching this subject, most notably on the Greedy Resistant instances. The model, even though its GR parameter value for this subject was zero, still failed to capture the means by which this subject was able to find optimal solutions in relatively few selections for these Greedy Resistant instance. Model II was not designed to learn, and this may be a reason for this poor fit. Participants in the Search condition, with the feedback available as a result of the specified solution size, had available to them information needed to learn that a strategy could fail. This could have resulted in the adoption of an alternate strategy: selecting vertices adjacent to high(est) degree vertices, in particular on those instances on which the GR strategy was designed to fail. The model, of course, was not able to learn or apply new strategies, and cannot be expected to model this behaviour. The fit by this measure, indicates that a model based on local strategies alone is not sufficient to capture the behaviour of all subjects. The subjects investigated in this Chapter, tasked with the Search version of the Vertex Cover problem were clearly using strategies other than those used by the model to find solutions. This supports the hypothesis that the feedback of the concrete goal can support strategy acquisition, and possibly even schema activation.

The manner by which the undos were implemented also is likely not representative of how subjects chose vertices to undo if unable to find a vertex cover that matched the goal's size. The model removed a proportion of vertices at random. Notably, to achieve the best fit, this parameter was set to one for both participants. However, both of these participants only undid some moves when they failed to find a cover of the appropriate size. Likely a more sophisticated strategy was used to determine which vertices to undo before trying again.

6.5 Summary

A number of attempts have been made to try to model human performance on problems shown to be computationally hard, in part to try to better understand what

processes or strategies might explain the often surprisingly close to optimal solutions found. Modelling work has focused to date on Optimization problems, and this work makes a first attempt to model human performance on not only the Optimization version, but also the Search version of a hard computational problem. Where previous work focused on Euclidean problems, most notably E-TSP, this work attempts to model human performance on a not-Euclidean problem. As a result, the global processing included in many models of E-TSP performance is excluded from these models of human performance on the Vertex Cover problem. The proposed models test the theory that human performance on two versions of a hard computational problem, the Vertex Cover problem, can be explained using only simple local strategies. The following picture emerges from these findings. The models' fits in terms of solution size and vertices found in the final solutions indicate that human performance can be explained in part by simple local strategies. However, the relatively poorer fit in terms of number of moves suggests that if these strategies are being applied by participants, then they certainly are not being applied with unchanging probabilities. Participants' ability to find good solutions in fewer moves than the models on select instances indicates that they are able to reject or avoid strategies when they are not applicable. At the same time, the greater amount of search needed by participants on other instances indicates that if the identified strategies are being applied that they are not applied as consistently as they are by the model. It is likely worth investigating what kinds of undo strategies might be identified by participants, as the naive random approach failed to capture how participants chose vertices to remove from the cover.

This attempt at modelling had the additional benefit of the discovery of the Toggling behaviour. This behaviour indicates that search for a solution to some of the provided instances may have pushed the limit of working memory, causing participants to try to leverage cognitive support provided by the interface to help make move selections. Notably, behaviour of this kind has never been reported in experimental studies of human performance on hard computational problems.

Chapter 7

Conclusions and Future Work

Due to this studys interdisciplinary nature, its main contribution is to bridge the gap between the fields of cognitive science and computational complexity, where they meet in the study of human problem solving. This kind of bridging work is important in particular when bringing together two fields of study like these, disciplines with very different origins and methodologies, but which have a great deal to contribute to each other on this, and many other, topics.

In addition to this bridging work, this research contributes specifically to the study of human problem solving in three main ways. It further refines our understanding of how people are able to cope with complexity, in particular when it may not be possible to know when a problem is solved. Second, it proposes a different approach for evaluating how people cope with complexity when they are free from being confounded by the challenge of determining when a solution is correct. Finally, the research study and computational model presented in Chapters 4, 5, and 6 expands our understanding of how humans cope with complexity, presenting novel performance results for human problem solving of hard computational problems.

7.1 Refinement of Terminology

A lot of territory was covered in Chapter 2 and 3, exploring and highlighting relevant problem-solving theory from both the fields of Cognitive Science and Computational Complexity. To review, this work has contributed to the improvement of the study of human problem solving of hard computational problems by refining shared terminology. In particular, the following issues were brought to light:

- The importance of differentiating between solving a problem and solving an instance of a problem, which is closely related to differentiating between finding an algorithm that solves a problem in general (as is typical in computer science, and upon which computational complexity results are typically based) and finding a solution to one or more instances of a problem (as is typical in Human Problem Solving, and upon which human performance results are typically based).

- The need to differentiate between factors that contribute to the subjective and objective difficulty of a problem-solving task. Subjective difficulty has been attributed to a number of factors including instance size, goal specificity, instance selection and order, and schema acquisition. Subjective difficulty stems from the interaction between the problem solver and instances of the problem. Objective difficulty, on the other hand, is based on the computational complexity of finding a general algorithm to solve a problem. Objective difficulty directly depends on the computational task at hand, independent of the problem solver's experience or expertise.

- The importance of understanding, at a computational level, what problem is being solved and not assuming that the problem being solved is the task given to problem solvers. This amounts to determining whether or not it is possible, given a problem formulation, for a problem to be encoded in the problem solver's internal representation. If a problem resists encoding, then effort must turn to determining what problem(s) might instead be being encoded, and therefore solved.

In light of these issues, some suggestions are now made as how best to move forward in the study of human problem solving when hard computational problems are to be used as instruments.

- Be cautious with the language used when discussing human performance results, and (correctly) refer to *solving an instance of a problem*, unless the subject has indeed found an algorithm that solves the problem in general (for all instances) in which case the phrase *solving a problem* is appropriate.

- Be cautious when making claims about the quality of a subject's solution framed in the context of the computational complexity of the problem in general. Instances of a problem, even an NP-Complete problem, can be easier (in the sense

of the complexity of the problem) or subjectively more difficult than the problem is in general. A good example that highlights this issue is how subjects can be *tricked* into finding very suboptimal tours to E-TSP instances, as a result of the Gestalt principle of good continuation [106]. Similarly, while no clear impact of instance order on performance found in this study, it is possible that instance order can result in differences in solution quality (as with the Einstellung effect [51, 62, 103]). Even though computational complexity theory can inform us about the objective difficulty of a problem in general, the interaction between the problem solver and instances of the problem (and other problems) contributes to the subjective difficulty of the task as well.

- Take care to determine as much as possible what problem is being solved by subjects at the *computational level*. As seen in this work, ill-defined aspects of a problem can interfere with a problem solver's ability to encode the problem, potentially rendering the presumed computational complexity of the given task irrelevant.

A direct consequence of the refinement of these concepts is the need to re-evaluate previous human performance results in experiments that used hard optimization problems as instruments. It also leads to two main issues in the design of future studies of this kind.

7.2 Designing Studies to Evaluate How People Cope with Complexity

The main contributions of this work speak to improving the design of studies of human performance on hard computation problems and improving how performance results are interpreted on past and current studies that use these hard computational problems as instruments. Foundational to this contribution is the notion of gaining a sound understanding of how problem formulation impacts what problem can be encoded in the problem solver's internal representation, which in turn directly impacts what problem can, in fact, be solved.

7.2.1 Coping with Ill-Defined Goals

Tasking problem solvers with instances of hard optimization problems has the potential to result in problem solvers being unable to encode the goal as given. Because the problem encoded in the system defines the problem being solved, an inability to encode the given task implies that a different problem is being solved. This in turn has important ramifications when attempting to interpret the results of these studies. For one, the complexity results of the original task may no longer be representative of the complexity results of the problem being solved. In fact, if the results of this study are representative of how this mechanism manifests itself, the encoded problem is likely easier than intended. This also has implications when attempting to model performance. As shown in Chapter 6, it can be possible to model human performance very closely if the correct goal modification is selected.

In Chapter 3, a framework is provided for decomposing a problem's goal to identify what aspect of the goal is ill-defined, thereby making it possible to find candidate problems which might be encoded in place of the given optimization task. In Chapter 4 this framework is used, to serve as an example, to decompose the goal of the optimization version of the two problems used in this study. These candidate modifications are evaluated to determine what problems best explain performance.

How does goal modification take place?

This research made no attempt to identify the mechanisms by which goal modification might take place, and instead focused on the problem at the computational level. But the question of what kinds of mechanisms might explain this process is clearly interesting, and deserving of investigation. It is very likely that goal modification as described here will be dependent on the nature of the task. For well-structured and otherwise well-defined problems like the computational problems typically found in the field of computational complexity, it may be possible, as done here, to decompose the problem's goals and identify candidate modifications. This is particularly true when working in a lab environment with subjects who are naive to the problem at hand. For real-world problems, however, the task of goal modification becomes more challenging, as complex factors like domain-specific knowledge and personal experience will impact how ill-defined goals are rendered well-defined.

Problem solvers do not appear to be aware of goal modification taking place, nor do they spend a great deal of time re-encoding the problem, suggesting that the un-

derlying mechanisms that explain this process are *fast and frugal* [67]. Furthermore, there is no reason to assume that goal modification occurs only once for a given problem or instance of a problem, or that it manifests itself identically between different problem solvers. It is likely that problem solvers adjust the encoding of the problem as they gain experience with it.

As a result, we find ourselves in a sea of unknowns if we wish to continue to use hard optimization problems as instruments in the study of human problem solving. Tasked with a problem with an ill-defined goal, problem solvers encode *some* problem, and may re-encode it *any number of times*, and experience and domain-specific knowledge will likely impact the types of modifications that can take place. All that is known is that given a task with an ill-defined goal, the problem solver *cannot* be solving the given task, in general, which has a number of implications.

What are the implications of goal modification?

If computational complexity is taken into account, then the interpretation of human performance results is dependent on the problem being solved. This issue was not addressed in previous studies of human performance on hard optimization problems, which opens up a new way of interpreting the results of a number of studies. In Chapter 3, E-TSP and N-Puzzle were decomposed to isolate the component of the goal that is ill-defined; however, no candidate modifications were proposed. This is a great starting point from which to investigate what goal modifications might best explain performance on these problems, and to review previous attempts to model this performance.

What are the implications of these findings in general?

The modification of ill-defined aspects of a problem has implications in other areas of problem-solving research and is not limited to the study of hard computational problems. For example, this approach is equally applicable in the study of decision-making or insight problems, and is closely related to explanations of performance in these areas. The difficulty of finding a solution to the Nine Dot problem, for example, has been attributed to problem solvers' inability to determine what constitutes a legal operator, rendering the set of legal operators ill-defined. Similarly, satisficing theory in decision making attempts to explain how people find solutions in a task environment rife with unknowns and uncertainties, by replacing the concept of *op-*

timizing with that of finding a solution that is *satisfactory*. Both these approaches try to explain performance by suggesting that problem solvers modify an aspect of the given problem. The difference in what I am presenting here, is that in addition to a proposed modification, this work recognizes that the modification of any aspect of the given problem can, and often does, result in a modification of the task. This acknowledgement of the implications of goal (or other problem aspect) modification *at the computational level*, makes it possible to better understand and evaluate algorithmic explanations for performance. There is potential to apply this same technique in other problem-solving research areas where ill-defined problem aspects are found.

It is worth noting, at this point, the difference between the performance impact of goal specificity, described earlier in this work, and the difference in performance based on the well-definedness of the goal. In maze tracing experiments, better performance was observed when the goal was not specified than when it was [104]. It is important to note, however, that the goal in both conditions of the maze-tracing experiment was well-defined. Problem solvers could tell when the goal had been attained. In the research done in this work, in contrast, the non-specified goal of optimization renders the goal ill-defined, and therefore problem solvers are unable to determine if the goal has been attained. Therefore, while goal specificity was found previously to negatively impact problem solving in the context of maze tracing, it is not directly related to the performance results presented in this work.

7.2.2 Coping with Complexity in Problems with Well-Defined Goals

A second main contribution of this work is the observation that it is possible to evaluate human performance on hard computational problems without the confounding factor of an ill-defined goal. As shown in Chapter 3, the Search version of hard optimization problems can be used to study how people navigate large problem spaces of hard problems, but with an encodable goal.

What are the implications of these findings in general?

If the results of this study are representative of a general pattern in problem solving, then the better performance observed on instances of the Search versions of the problems used in this research implies that people may in fact be better at finding solutions to instances of hard computational problems than previously thought. This

is a surprising result, one which I did not expect, and one which I think warrants further investigation. When I first became aware of human performance results on hard optimization problems, I assumed that the surprisingly good performance results might be more an artifact of study design than indication of how good people are at coping with complexity. While this work does not discount this possibility, instance size and selection may have unintentionally resulted in an easier task than desired; performance on instances of the Search version problems used in this study is still better than that on those same instances of the associated Optimization version. This raises the possibility that similar results might be found on other problems like E-TSP or N-Puzzle.

The use of a version of a problem with an encodable goal has the added advantage of having the potential to reinforce strategy and/or schema acquisition. While the feedback from the goal in this study was not sufficient to support strategy or schema acquisition, this study was also not purposely designed with this in mind. This opens the possibility of designing studies using search (or decision) versions of hard computational problems to investigate whether or not novices are able to acquire strategies or schemata.

Another observation that was made in Chapter 5, above, is that the great deal of search that participants in the Search condition of both problems used in this study indicates a deep level of engagement with what are objectively complex problem-solving tasks. This has important educational, workplace, and tool-design implications. As Polya stated, a key component in teaching problem solving is getting learners to engage with hard problems [78]. If the results of this work are representative of the kind of engagement that can be elicited in naive problem solvers on these hard computational tasks, then visually presented tasks like these might prove to be useful tools in teaching problem solving. In the workplace and in the design of tools used for solving complex tasks, the use of concrete, encodable goals could also serve to encourage engagement with complex tasks. While it is known that data representation in general impacts problem-solving performance [12, 60, 69, 71], the well-definedness of problem goals also clearly impacts performance. This knowledge can be leveraged to improve workplace patterns, by replacing ill-defined goals with well-defined alternatives to better represent complex tasks such that they can be meaningfully encoded in order to result in better engagement..

7.3 Performance Results

Another main contribution of this research study has been to further expand our understanding of the power of human problem solving. In addition to discovering that human performance on two new hard optimization problems is in keeping with previous findings using other hard optimization problems, this work presents novel performance results for human problem solving of hard computational problems. This study was the first of its kind, not only in investigating performance on the Search version of two NP-Complete problems, but also in comparing this performance to that on the associated Optimization version of these problems. In addition, this work presents for the first time human performance results for a NP-Complete maximization problem, the Independent Set problem. In this section, we review briefly the highlights of these findings.

- Tasked with instances of Vertex Cover and Independent set, two hard optimization problems, humans are able to find close to optimal solutions in relatively few moves. These findings are in keeping with previous research using hard optimization problems as instruments.

- Despite being able to find close to optimal solutions when tasked with hard optimization problems, participants are not able to find very many optimal solutions, which is also in keeping with previous research using hard optimization problems as instruments.

- People are able to find many more optimal solutions if tasked with the Search version of a hard computational problem than if tasked with its Optimization version. This is somewhat surprising and suggests that previously reported problem-solving performance on hard optimization problems may not be indicative of how well people are able to do on these hard problems.

- It appears that minimization may not be easier than maximization in the context of problem solving hard computational problems.

7.4 Unanswered Questions

The results of this research suggest that there is more work to be done in the area of studying human performance on hard computational problems. In addition to the

future work described above, the following questions come to mind:

- Do the results found here apply to the Search (or Decision) version of other known hard optimization problems? If so, human performance results on the Search version of E-TSP might be better than that found for the optimization version in previous research.

- Is the better performance on the maximization problem observed in this study representative of an inherent difference between performance on maximization versus minimization problems? Or is it a result of some other aspect of the problems, or instances, used in this study?

- What role does cognitive support play in problem solving of hard computational problems? Tasking people with the Search version of both Vertex Cover and Independent Set resulted in more cognitive load than tasking them with the Optimization versions of these problems. The Toggling behaviour observed predominantly in the Search condition was interpreted to be subjects using the interface to gain information about either the current or possible subsequent states of the graph. This raises the following related questions:

    1. What kinds of cognitive support might be useful to participants when tasked with hard problems like these?
    2. How might cognitive support impact performance on hard computational problems?

- What role does problem representation play in human performance on hard computational problems? In this work, like most previous studies of human performance on hard computational problems, subjects were presented graph representations of a task. Even though Vertex Cover, Independent Set, and E-TSP lend themselves quite naturally to this representation, they can equally be represented in other ways. Graphs appear to be a natural way to represent relational information, and the quality of human performance on visually presented graph problems may be dependent upon this representation. This could be tested using alternate representations instead. Graphs can be represented, for example, as an adjacency matrix without loss of information.

- How does performance on hard computational problems differ between experts and novices? Subjects in this study were purposely chosen to be naive to the

hard computational problems used. It is likely that experience with computational complexity influences the kinds of goal modifications that would result when tasked with hard optimization problems.

- Do people optimize when tasked with tractable computational problems? Problems in the Class P, like the Minimum Spanning Tree problem, can also be phrased as optimization problems. Being computationally tractable does not render their goal encodable, and it is interesting to wonder how goal modification might differ if the underlying problem is not computationally hard.

- How might participants perform if tasked with the candidate modifications suggested for the optimization versions of the Vertex Cover and Independent Set problems used in this study? This might help support or refute the hypothesis that these modifications are representative of how the goal is encoded.

While these questions are not exhaustive, they serve to indicate how much more we have to learn about the factors that contribute to how humans are able to cope with complexity.

7.5 Final Thoughts

This research has taken important steps to extend our understanding of how humans cope with complexity, by examining performance on problems other than the infamous E-TSP. While the study is not unique in this, it is still only one of a handful of research projects that have ventured away from TSP variants in the pursuit of an understanding of how people cope with complexity. More importantly, however, this research represents a first investigation of human performance on the search version of problems, which resulted in important insight about how humans cope with not only hard problems, but problems for which a solution cannot be consistently verified. Finally, I hope that the lessons learned in this research will serve to improve the design of research studies to come, as well as our understanding of the power of human problem solving.

Bibliography

[1] The R Project for Statistical Computing. Accessed: 2015-0507.

[2] Greg Aloupis, Erik D Demaine, and Alan Guo. Classic Nintendo Games are (NP-)Hard. *CoRR*, pages 1–21, 2012.

[3] Helmut Alt, Hans L Bodlaender, Marc van Kreveld, Günter Rote, and Gerard Tel. Wooden Geometric Puzzles: Design and Hardness Proofs. In *4th international conference on Fun with algorithms*, pages 16–29, 2007.

[4] John R Anderson. Acquisition of Cognitive Skill. *Psychological Methods*, 89:369–406, February 1982.

[5] Sanjeev Arora and Boaz Barak. *Computational Complexity*. A Modern Approach. Cambridge Univerisity Press, 2007.

[6] Ivan K Ash and Jennifer Wiley. The nature of restructuring in insight: An individual-differences approach. *Psychonomic Bullitin & Review*, 13:66–73, November 2006.

[7] Michael E Atwood, Michael E J Masson, and Peter G Polson. Further explorations with a process model for water jug problems. *Memory & Cognition*, 8(2):182–192, November 1980.

[8] Michael E Atwood and Peter G Polson. A Process Model for Water Jug Problems. *Cognitive Psychology*, 8:191–216, February 1976.

[9] Alan D Baddeley and Graham Hitch. Working Memory. In *Psychology of Learning and Motivation*, pages 47–89. Elsevier, 1974.

[10] Bradley Best and Herbert Simon. Simulating Human Performance on the Traveling Salesman Problem. *Proceedings of the third international conference on cognitive modeling*, pages 1–6, 2003.

[11] Nicholas R Burns, Michael D Lee, and Douglas Vickers. Are Individual Differences in Performance on Perceptual and Cognitive Optimization Problems Determined by General Intelligence? *Journal of Problem Solving*, pages 1–15, January 2006.

[12] John M Carroll, John C Thomas, and Ashok Malhotra. Presentation and representation in design problem-solving. *British Journal of Psychology*, 71:143–153, December 1980.

[13] Sarah Carruthers, Michael Masson, and Ulrike Stege. Human Performance on Hard Non-Euclidean Graph Problems: Vertex Cover. *Journal of Problem Solving*, 5(1):34–55, 2012.

[14] Sarah Carruthers, Todd Milford, Timothy Pelton, and Ulrike Stege. Draw a Social Network. In *the 16th annual joint conference*, pages 178–182, New York, New York, USA, 2011. ACM Press.

[15] Jason M Chein, Robert W Weisberg, Naomi L Streeter, and Shaleigh Kwok. Working memory and insight in the nine-dot problem. *Memory & Cognition*, 38(7):883–892, October 2010.

[16] Zhe Chen and Lei Mo. Schema Induction in Problem Solving: A Multidimensional Analysis. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 30(3):583–600, 2004.

[17] Michelene T H Chi and Robert Glaser. Problem-Solving Ability. pages 1–27, July 2007.

[18] Edward P Chronicle, Thomas C Ormerod, and James N MacGregor. When insight just won't come: The failure of visual cues in the nine-dot problem. *The Quarterly Journal of Experimental Psychology A*, 54(3):903–919, August 2001.

[19] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[20] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2001.

[21] Janet E Davidson. Insights about insightful problem solving. In *The Psychology of Problem Solving*, pages 150–171. Cambridge Univerisity Press, May 2003.

[22] Erik D Demaine, Susan Hohenberger, and David Liben-Nowell. Tetris is Hard, Even to Approximate. pages 351–363. Computing and Combinatorics, 2008.

[23] Rod G Downey and Michael Ralph Fellows. *Parameterized complexity*, volume 3. springer New York, 1999.

[24] Matthew Dry, Michael D Lee, Douglas Vickers, and Peter Hughes. Human Performance on Visually Presented Traveling Salesperson Problems with Varying Numbers of Nodes. *Journal of Problem Solving*, 1(1):20–32, 2006.

[25] Matthew Dry, Kym Preiss, and Johan Wagemans. Clustering, Randomness, and Regularity: Spatial Distributions and Human Performance on the Traveling Salesperson Problem and Minimum Spanning Tree Problem. *Journal of Problem Solving*, 4(1):1–17, 2012.

[26] Matthew J Dry and Elizabeth L Fontaine. Fast and Efficient Discrimination of Traveling Salesperson Problem Stimulus Difficulty. *The Journal of Problem Solving*, 7(1), November 2014.

[27] Karl Duncker. On Problem-Solving. *Psychological Monographs*, 58(5):1–120, November 1945.

[28] Dennis E Egan and James G Greeno. Theory of rule induction: Knowledge acquired in concept learning, serial pattern learning, and problem solving. 1974.

[29] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[30] Gerd Gigerenzer and Wolfgang Gaissmaier. Heuristic Decision Making. *The Annual Review of Psychology*, 62(1):451–482, January 2011.

[31] Gerd Gigerenzer and Daniel G Goldstein. Mind as Computer. *Creativity Research Journal*, 9(2 & 3):131–144, 1996.

[32] Scott M Graham, Anupam Joshi, and Zygmunt Pizlo. The traveling salesman problem: A hierarchical model. *Memory & Cognition*, 28(7):1191–1204, 2000.

[33] James G Greeno. Hobbits and Orcs: Acquisition of a Sequential Concept. *Cognitive Psychology*, 6:270–292, September 1974.

[34] James G Greeno and Herbert A Simon. Problem Solving and Reasoning. Technical Report 85, Carnagie Mellon University, Pittsburgh, PA, October 1988.

[35] Yll Haxhimusa, Edward Carpenter, Joseph Catrambone, David Foldes, Emil Stefanov, Laura Arns, and Zygmunt Pizlo. 2D and 3D Traveling Salesman Problem. *Journal of Problem Solving*, 3(2), September 2011.

[36] Yll Haxhimusa, W G Kropatsch, Zygmunt Pizlo, A Ion, and A Lehrbaum. Approximating TSP Solution by MST based Graph Pyramid. In *6th IAPR-TC-15 international conference on Graph-based representations in pattern recognition*, pages 295–306, December 2007.

[37] Mercedes Hidalgo-Herrero, Pablo Rabanal, Ismael Rodríguez, and Fernando Rubio. Comparing Problem Solving Strategies for NP-hard Optimization Problems. *Fundamenta Informaticae*, 124:1–25, September 2013.

[38] T. Horgan and J. Tienson. *Connectionism and the philosophy of psychology.* Cambridge, MA, MIT Press, 1996.

[39] Robin Jeffries, Peter G Polson, and Lydia Razran. A Process Model for Missionaries-Cannibals and Other River-Crossing Problems. *Cognitive Psychology*, 9:412–440, September 1977.

[40] Viggo Kann. *On the approximability of NP-complete optimization problems.* PhD thesis, Royal Institute of Technology Stockholm, 1992.

[41] Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for tsp. In *Algorithms and Computation*, pages 568–578. Springer, 2013.

[42] Graham Kendall, Andrew Parkes, and Kristian Spoerer. A Survey of NP-Complete Puzzles. *ICGA Journal*, pages 13–34, September 2012.

[43] Trina C Kershaw and Stellan Ohlsson. Multiple Causes of Difficulty in Insight: The Case of the Nine-Dot Problem. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 30(1):3–13, 2004.

[44] William Kocay and Donald L Kreher. *Graphs, algorithms, and optimization*. Chapman & Hall/CRC, 2004.

[45] Xiaohui Kong and Christian D Schunn. Global vs. local information processing in visual/spatial problem solving: The case of traveling salesman problem. *Cognitive Systems Research*, 8(3):192–207, September 2007.

[46] Kenneth Kotovsky. Why Are Some Problems Hard? Evidence from Tower of Hanoi. *Cognitive Psychology*, 17:248–294, September 1985.

[47] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[48] Gilbert Laporte, Ardavan Asef-Vaziri, and Chelliah Sriskandarajah. Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, pages 1461–1467, 1996.

[49] Jill Larkin, John McDermott, Dorothea P Simon, and Herbert A Simon. Expert and Novice Performance in Solving Physics Problems. *Science*, 208:1–9, June 1980.

[50] Chen-Chung Liu, Yuan-Bang Cheng, and Chia-Wen Huang. Computers & Education. *Computers & Education*, 57(3):1907–1918, November 2011.

[51] Abraham S Luchins. Mechanization in Problem Solving. *Psychological Monographs*, 54(6):1–102, November 1942.

[52] James N MacGregor, Edward P Chronicle, and Thomas C Ormerod. Convex hull or crossing avoidance? Solution heuristics in the traveling salesperson problem. *Memory & Cognition*, 32(2):260–270, 2004.

[53] James N MacGregor, Edward P Chronicle, and Thomas C Ormerod. A Comparison of Heuristic and Human Performance on Open Versions of the Traveling Salesperson Problem. *Journal of Problem Solving*, 1(1):33–43, October 2008.

[54] James N MacGregor and Thomas C Ormerod. Human performance on the traveling salesman problem. *Perception & Psychophysics*, 58(4):527–539, 1996.

[55] James N MacGregor, Thomas C Ormerod, and Edward P Chronicle. Spatial and contextual factors in human performance on the travelling salesperson problem. *Perception*, 28(11):1417–1427, 1999.

[56] James N MacGregor, Thomas C Ormerod, and Edward P Chronicle. A model of human performance on the traveling salesperson problem. *Memory & Cognition*, 28(7):1183–1190, 2000.

[57] James N MacGregor, Thomas C Ormerod, and Edward P Chronicle. Information processing and insight: A process model of performance on the nine-dot and related problems. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 27(1):176–201, 2001.

[58] Norman R F Maier. Reasoning in Humans. *Journal of comparative psychology*, 10(2):115–143, November 1930.

[59] Norman R F Maier. Reasoning in Humans III. *Journal of Experimental Psychology*, 35(5):349–360, November 1945.

[60] R Marfil, F Escolano, and A Bandera. Graph-Based Representations in Pattern Recognition and Computational Intelligence. *IWANN*, pages 399–406, October 2009.

[61] D. Marr. *Vision: A computational approach*. Freeman & Co., San Francisco, 1982.

[62] Robert F Mawer and John Sweller. Effects of Subgoal Density and Location on Learning During Problem Solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 8(3):252–259, February 1982.

[63] E. Millgram. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[64] Richard D Morey, Jeffrey N Rouder, Tahira Jamil, and Maintainer Richard D Morey. Package bayesfactor. 2014.

[65] Allen Newell and Herbert A Simon. Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristic. In *Western Computer Proceedings*, pages 218–230, September 1957.

[66] Allen Newell and Herbert A Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, New Jersey, 1 edition, 1972.

[67] Ben R Newell, Nicola J Weston, and David R Shanks. Empirical tests of a fast-and-frugal heuristic: Not everyone "takes-the-best". *Organizational Behavior and Human Decision Processes*, 91(1):82–96, May 2003.

[68] Raymond S Nickerson. Null Hypothesis Significance Testing: A Review of an Old and Continuing Controversy. *Psychological Methods*, 5(2):241–301, September 2000.

[69] Donald A Norman. *The design of everyday things*. Basic books, 2002.

[70] Donald A Norman and Daniel G Bobrow. On data-limited and resource-limited processes. *Cognitive psychology*, 7(1):44–64, 1975.

[71] Jan M Noyes and Kate J Garland. Solving the Tower of Hanoi: does mode of presentation matter? *Computers in Human Behavior*, 19(5):579–592, September 2003.

[72] Thomas C Ormerod and Edward P Chronicle. Global perceptual processing in problem solving: The case of the traveling salesperson. *Perception & Psychophysics*, 61(6):1227–1238, 1999.

[73] S E Palmer. *Vision science: photons to phenomenology*. The MIT Press Cambridge, 1999.

[74] Christos H Papadimitriou. The Euclidean Traveling Salesman Problem is NP-Complete. *Theoretical Computer Science*, 4:237–244, November 1977.

[75] Federico Pasin and Hélène Giroux. Computers & Education. *Computers & Education*, 57(1):1240–1254, August 2011.

[76] Zygmunt Pizlo and Zheng Li. Solving combinatorial problems: The 15-puzzle. *Memory & Cognition*, 33(6):1069–1084, 2005.

[77] Zygmunt Pizlo, Emil Stefanov, John Saalweachter, Zheng Li, Yll Haxhimusa, and Walter G Kropatsch. Traveling Salesman Problem: A Foveating Pyramid Model. *The Journal of Problem Solving*, 1(1), January 2006.

[78] George Polya. *How to solve it: A new aspect of mathematical method*. Princeton university press, 1945.

[79] Z.W. Pylyshyn. *Computation and Cognition: Towards a Foundation for Cognitive Science*. Cambridge, MA, MIT Press, 1984.

[80] Adrian E Raftery. Bayesian Model Selection in Social Research. *Sociological Methodology*, 25:111–163, March 1995.

[81] Daniel Ratner and Manfed Warmuth. Finding a Shortest Solution for the NxN Extension of the 15-Puzzle is Intractable. In *AAAI-86*, pages 168–172, September 1986.

[82] Alexander Reinefeld. Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*. *International Joint Conference on Artificial Intelligence*, pages 1–6, October 1993.

[83] Jeffrey N Rouder, Richard D Morey, Paul L Speckman, and Jordan M Province. Default Bayes factors for ANOVA designs. *Journal of Mathematical Psychology*, 56(5):356–374, October 2012.

[84] D.E. Rumelhart, J..K. MacClelland, and the PDP Research Group. *Parallel distributed processing. Explorations in the microstructure of cognition. Volume 1: Foundations*. Cambridge, MA, MIT Press, 1986.

[85] John Saalweachter and Zygmunt Pizlo. Non-Euclidean Traveling Salesman Problem. In T Kugler, J C Smith, T Connolly, and Sun Y-J, editors, *Decision modeling and behavior in complex and uncertain environments*, pages 339–358. Springer, New York, September 2008.

[86] Allan Scott. *On the Parameterized Complexity of Finding Short Winning Strategies in Combinatorial Games*. PhD thesis, University of Victoria, April 2009.

[87] Allan Scott and Ulrike Stege. Parameterized chess. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation*, volume 5018 of *Lecture Notes in Computer Science*, pages 172–189. Springer Berlin Heidelberg, 2008.

[88] Allan Scott, Ulrike Stege, and Iris van Rooij. Minesweeper May Not Be NP-Complete but Is Hard Nonetheless. *The Mathematical Intelligencer*, 33(4):5–17, November 2011.

[89] Merlijn Sevenster. Battleships as a decision problem. *ICGA Journal*, 27(3):142–149, 2004.

[90] M-Z Shieh and S-C Tsai. Jug measuring: Algorithms and complexity. *Theoretical Computer Science*, 396:50–62, 2008.

[91] Herbert A Simon. A Behavioral Model of Rational Choice. *The Quarterly Journal of Economics*, 69(1):99–118, February 1955.

[92] Herbert A Simon. Rational Choice and the Structure of the Environment. *Psychological Monographs*, 63:129–138, February 1956.

[93] Herbert A Simon. The Structure of Ill Structured Problems*. *Artificial Intelligence*, pages 1–21, October 1973.

[94] Herbert A Simon. INVARIANTS OF HUMAN BEHAVIOR. *Annual Review of Psychology*, 41:1–20, March 1990.

[95] Herbert A Simon. Artificial intelligence: an empirical science. *Artificial Intelligence*, 77(1):95–127, 1995.

[96] Herbert A Simon, George B Dantzig, Robin Hogarth, Charles R Plott, Howard Raiffa, Thomas C Schelling, Kenneth A Shepsle, Richard Thaler, Amos Tversky, and Sidney Winter. Decision Making and Problem Solving. *Interfaces*, 17(5):11–31, September 1987.

[97] Herbert A Simon and Glenn Lea. Problem Solving and Rule Induction: A Unified View. In Lee W Gregg, editor, *Knowledge and Cognition*, pages 105–128. Lawrence Erlbaum Associates, May 1973.

[98] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[99] Kurt Squire. Video Games in Education. *Int. J. Intell. Games & Simulation*, 2:1–16, May 2003.

[100] Jeff Stuckman and Guo-Qiang Zhang. Mastermind is NP-complete. *arXiv preprint cs/0512049*, 2005.

[101] John Sweller. The Effect of Task Complexity and Sequence on Rule Learning and Problem Solving. *British Journal of Psychology*, 67(4):553–558, December 1976.

[102] John Sweller. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12:257–285, October 1988.

[103] John Sweller and W Gee. Einstellung, the Sequence Effect, and Hypothesis Theory. *Journal of Experimental Psychology: Human Learning and Memory*, 4:513–526, February 1978.

[104] John Sweller and Marvin Levine. Effects of Goal Specificity on Means-Ends Analysis and Learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 8:463–474, February 1982.

[105] John Sweller, Robert F Mawer, and Mark R Ward. Development of Expertise in Mathematical Problem Solving. *Journal of Experimental Psychology*, 112:639–661, February 1983.

[106] Susanne Tak, Marco Plaisier, and Iris van Rooij. Some Tours are More Equal than Others:. *Journal of Problem Solving*, 2(1):4–28, October 2008.

[107] John C Thomas. An Analysis of Behavior in the Hobbits-Orcs Problem. *Cognitive Psychology*, 6:257–269, September 1974.

[108] Edward L Thorndike. Animal Intelligence. *The Psychological Review*, 2(4):1–113, November 1898.

[109] Peter M Todd and G Gigerenzer. Précis of Simple heuristics that make us smart. *Behavioral and Brain Sciences*, pages 727–780, September 2000.

[110] I. van Rooij. The Tractable Cognition Thesis. *Cognitive Science*, 32:939–984, 2008.

[111] Iris van Rooij. *Tractable Cognition: Complexity Theory in Cognitive Psychology*. PhD thesis, University of Victoria, University of Victoria, 2003.

[112] Iris van Rooij, Alissa Schactman, Helena Kadlec, and Ulrike Stege. Perceptual or Analytical Processing? Evidence from Children's and Adult's Performance on the Euclidean Traveling Salesperson Problem. *Journal of Problem Solving*, 1(1):44–73, 2006.

[113] Iris van Rooij, Alissa Schactman, and Ulrike Stege. Convex hull and tour crossings in the Euclidean traveling sales person problem: Implications for human performance studies. *Memory & Cognition*, 31(2):215–220, September 2003.

[114] Iris van Rooij, Ulrike Stege, and Alissa Schactman. Convex hull and tour crossings in the Euclidean traveling salesperson problem: Implications for human performance studies. *Memory & Cognition*, 31(2):215–220, 2003.

[115] Douglas Vickers, Marcus Butavicius, Michael D Lee, and Andrei Medvedev. Human performance on visually presented Traveling Salesman problems. *Psychological Research*, 65:34–45, February 2001.

[116] Amy L Walwyn and Daniel J Navarro. Minimal Paths in the City Block: Human Performance on Euclidean and Non-Euclidean Traveling Salesperson Problems. *Journal of Problem Solving*, 3(1):93–105, September 2011.

[117] Marilyn C Welsh, Trey Satterlee-Cartmell, and Michelle Stine. Towers of Hanoi and London: Contribution of Working Memory and Inhibition to Performance. *Brain and Cognition*, 41:231–242, October 1999.

[118] M Wertheimer. Laws of Organization in Perceptual Forms. In W Ellis, editor, *A source book of Gestalt psychology*, pages 71–88. London: Routledge & Kegan Paul, 1938.

Appendix A

Additional Information

A.1 Big-O Notation

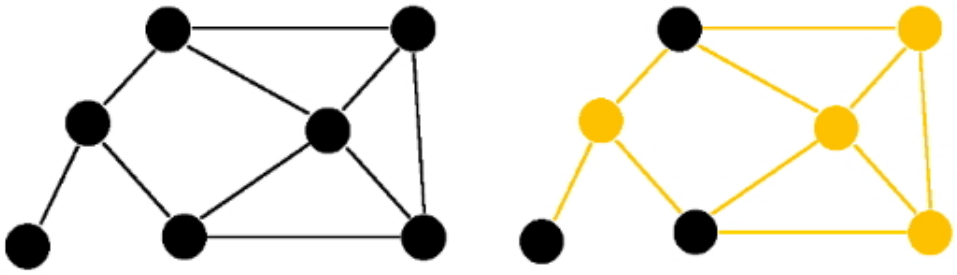
When considering the running time of an algorithm, it is often useful to argue about its *worst case* running time in terms of the input size. One way of doing to is to use *asymptotic analysis*. Given a function $f(n)$, we bound its growth by considering only its highest order term, also ignoring any constant factors. Consider, for example the function $f(n) = 5n^3 + 4n^2 + 27n + 1000$. The highest order term is $5n^3$, and therefore it is bounded, using *big-O* notation as follows: $f(n) = O(n^3)$.

More formally, given two functions $f, g : N \rightarrow R^+$, we say that $f(n) = O(g(n))$ if there exist positive integers c, d, such that $f(n) \leq cg(n)$ for all $n \geq d$ [98].

A.2 Participant Instructions

The Guard Problem:

Find the fewest guards necessary to protect all the art in the corridors of each art gallery. You will be presented drawing of the layout of an art gallery. The black circles represent guard-posts, and lines represent corridors containing priceless art. Your task is to place enough guards at the guard-posts so that every corridor can be viewed by at least one guard.
On the left is an example art gallery layout. A guard positioned at a post can view any of the corridors connected to that post. On the right is an example solution. The yellow circles represent guardposts with guards.



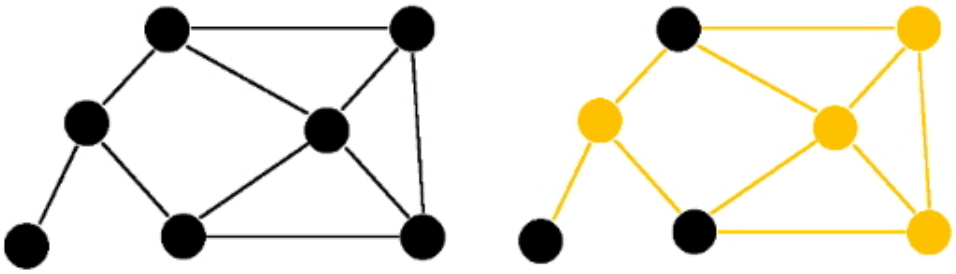
A gallery with 7 guardposts and 10 hallways

An optimal solution with 4 guards

Figure A.1: Vertex Cover Optimization condition instructions.

The Guard Problem:

Can you find a way to place a specified number of guards to protect the art in the corridors of the art gallery?
On each page is a drawing of the layout of an art gallery, and a GOAL number of guards. The black circles represent guard-posts, and lines represent corridors containing priceless art. Your task is to try to place the GOAL number of guards at the guard-posts so that every corridor can be viewed by at least one guard. A guard positioned at a post can view any of the corridors connected to that post.
On the left is an example art gallery layout. On the right is an example solution. The yellow circles represent guardposts with guards.
If it is not possible, please type: NO, in the Goal Possible field.



A gallery with 7 guardposts and 10 hallways;

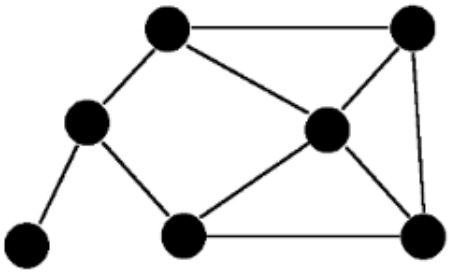
A solution with 4 guards

Figure A.2: Vertex Cover Search condition instructions.

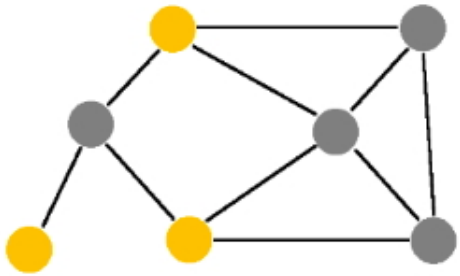
The Independent People Problem:

Find the largest group of people in a social network who don't know each other.

On each page is a drawing of a social network.
The circles represent people, and lines represent friendships or relationships between these people. Your task is to find the biggest set of people that don't know each other: people who are independent (that is, not directly connected to one another). Independent people may or may not have friends in common.
On the left is an example social network. On the right is an example solution. The yellow circles are a group of independent people. The grey circles are people who cannot be added to the group of independent people because they are connected to people in the group.



A problem with 7 people connected by 10 relationships

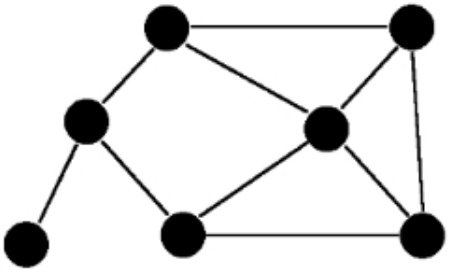


An optimal solution with 3 independent people

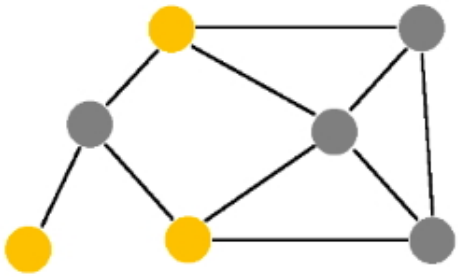
Figure A.3: Independent Set Optimization condition instructions.

The Independent People Problem:

Find a group of a specified number of people in a social network who don't know each other.
On each page is a drawing of a social network.
The circles represent people, and lines represent friendships or relationships between these people. Your task is to find a specific number of people that don't know each other: people who are independent (that is, not directly connected to one another). Independent people may or may not have friends in common.
On the left is an example social network. On the right is an example solution. The yellow circles are a group of independent people. The grey circles are people who cannot be added to the group of independent people because they are connected to people in the group.
If it is not possible, please type: NO, in the Goal Possible field.



A problem with 7 people connected by 10 relationships



A solution with 3 independent people

Figure A.4: Independent Set Search condition instructions.

A.3 Graphs Used in Study

In this section, the graphs used for the study are presented. Note that the labelling of vertices (letters on or beside the vertices) was not included in the versions given to participants, and are included here for reference. D1 graphs contain D1 vertices at onset. D2 graphs contain D2 vertices at onset. Greedy Resistant (GR) graphs are resistant to the Greedy strategy (Vertex Cover conditions only). No Strategy (NS) graphs contain no D1 or D2 vertices at onset, and are not resistant to the Greedy strategy. Random graphs were generated by adding edges at random.

Table A.1: Summary of graph properties. n is the number of vertices, e is the number of edges, D_{MAX} is the degree of the highest degree vertex, W is the longest of all shortest paths between all pairs of vertices, OPT_{VC} is the size of a minimum vertex cover, OPT_{IS} is the size of a maximum independent set, and Graph Type is the type of graph as described above

Graph ID	n	e	D_{MAX}	W	OPT_{VC}	OPT_{IS}	**Graph Type**	**Graph Size**
0	15	18	6	6	9	6	D1	small
1	15	20	6	6	9	6	D1	small
2	23	27	6	12	14	9	D1	medium
3	26	31	7	8	16	10	D1	medium
4	36	43	5	15	19	17	D1	large
5	14	19	5	5	7	7	D2	small
6	18	22	4	10	9	9	D2	small
7	27	36	5	15	12	15	D2	medium
8	28	37	5	11	12	16	D2	medium
9	36	74	7	18	15	21	D2	large
10	13	18	6	4	7	6	GR	small
11	19	36	8	15	8	11	GR	small
12	22	32	5	8	12	10	GR	medium
13	25	37	5	8	15	10	GR	medium
14	43	62	7	12	23	20	GR	large
15	16	19	4	7	8	8	NS	small
16	11	19	5	5	6	5	NS	small
17	21	25	3	8	10	11	NS	medium
18	25	37	4	12	12	13	NS	medium
19	39	58	6	16	18	21	NS	large
20	16	23	6	7	9	7	Random	small
21	16	26	7	11	9	7	Random	small
22	25	41	7	9	12	13	Random	medium
23	25	39	6	16	12	13	Random	medium
24	36	58	9	16	18	18	Random	large

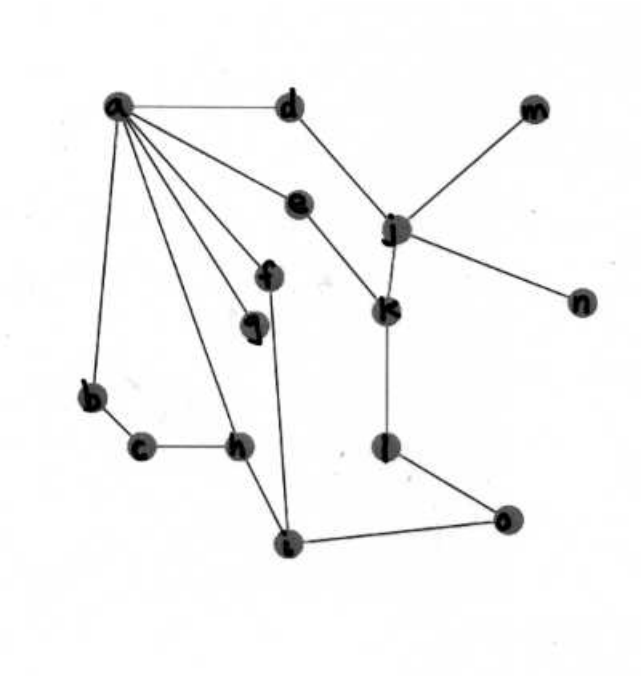


Figure A.5: Graph 0, a D1 graph

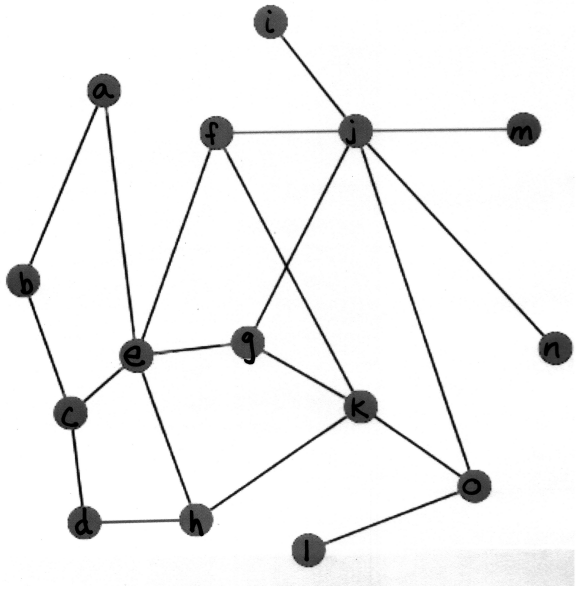


Figure A.6: Graph 1, a D1 graph

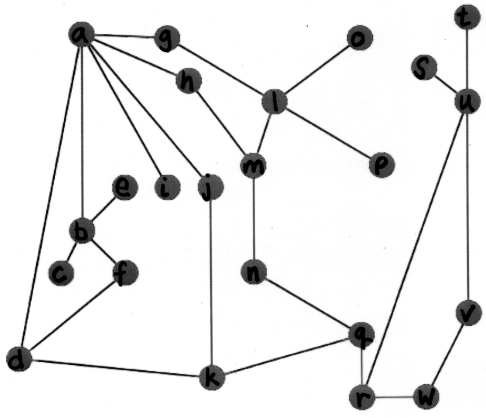


Figure A.7: Graph 2, a D1 graph

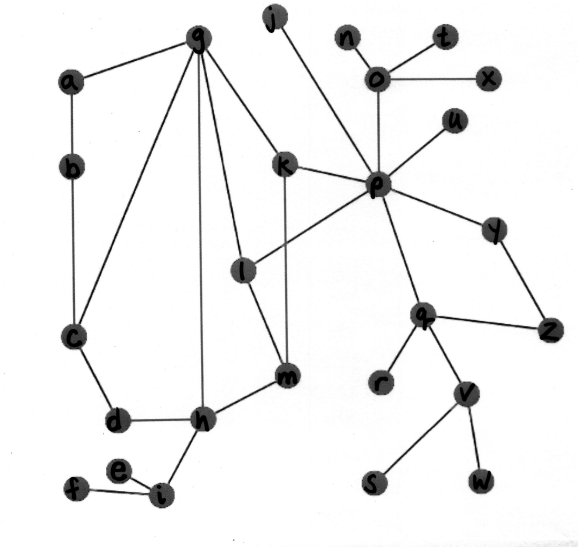


Figure A.8: Graph 3, a D1 graph

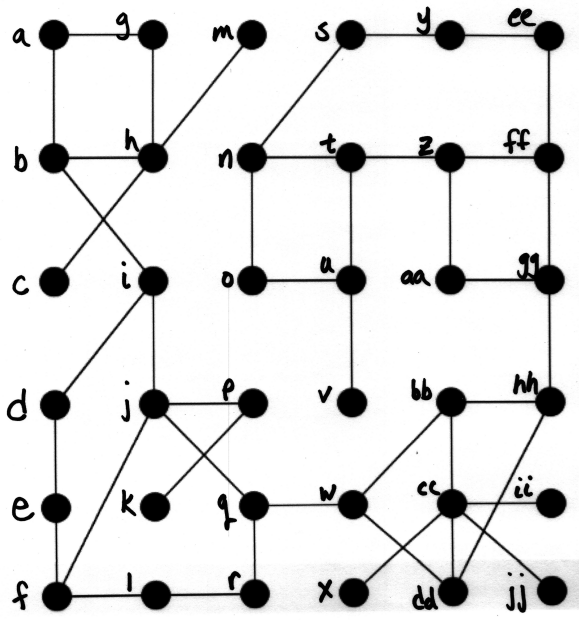


Figure A.9: Graph 4, a D1 graph

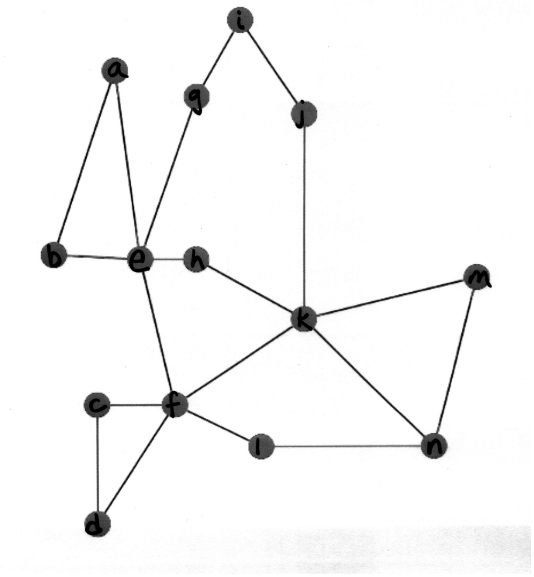


Figure A.10: Graph 5, a D2 graph

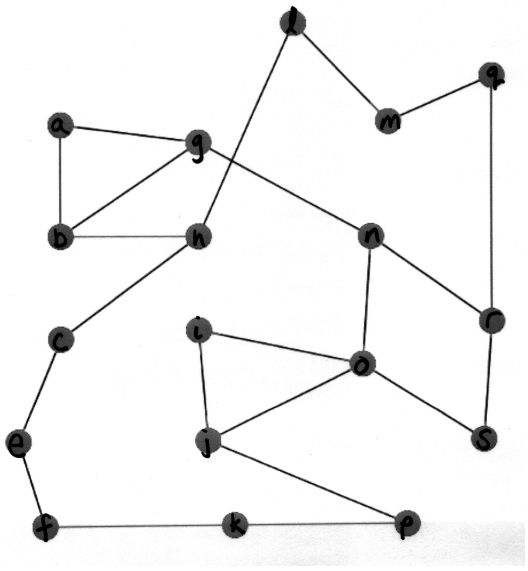


Figure A.11: Graph 6, a D2 graph

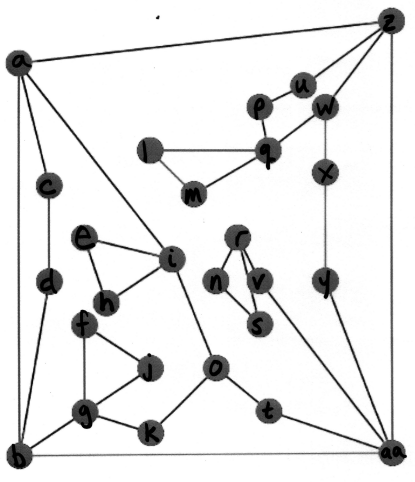


Figure A.12: Graph 7, a D2 graph

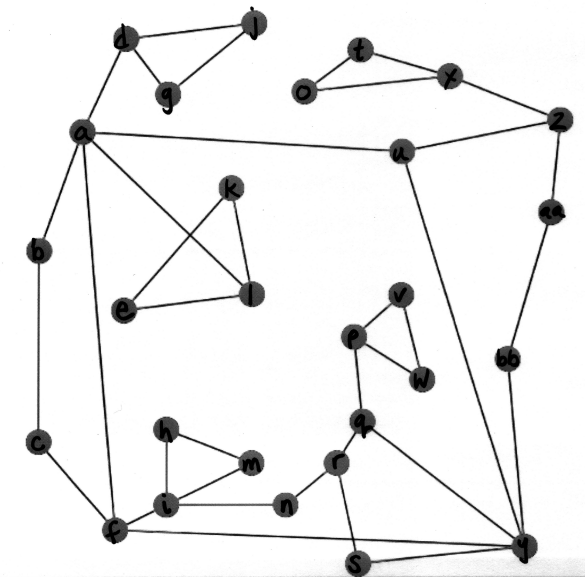


Figure A.13: Graph 8, a D2 graph

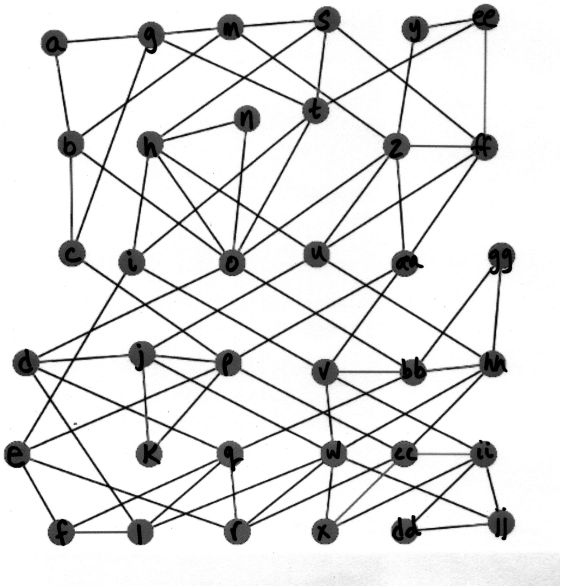


Figure A.14: Graph 9, a D2 graph

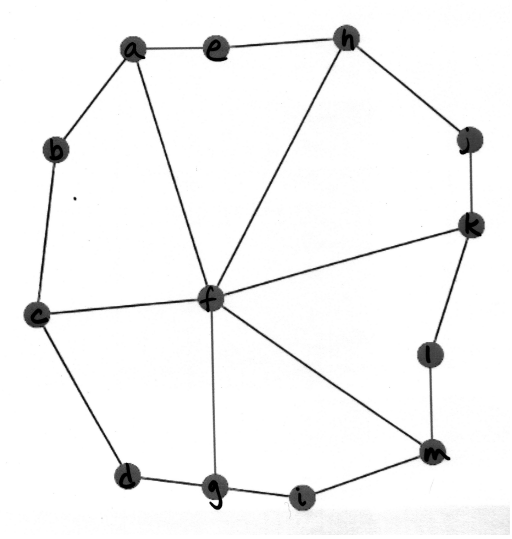


Figure A.15: Graph 10, a Greedy Resistant graph

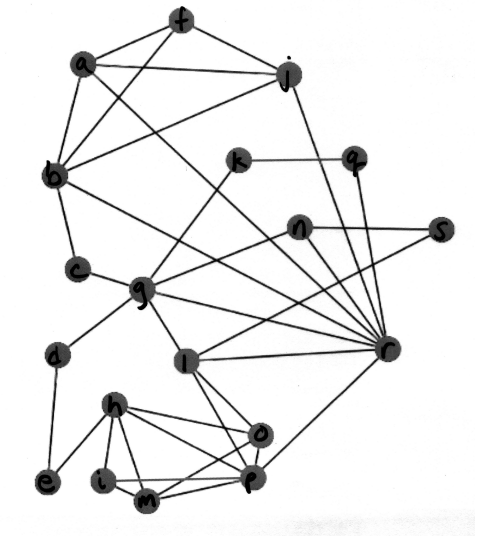


Figure A.16: Graph 11, a Greedy Resistant graph

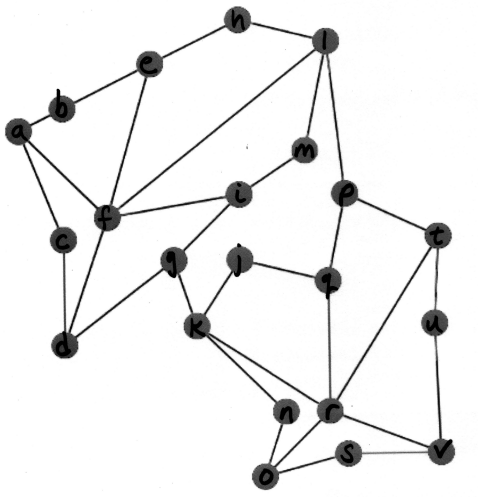


Figure A.17: Graph 12, a Greedy Resistant graph

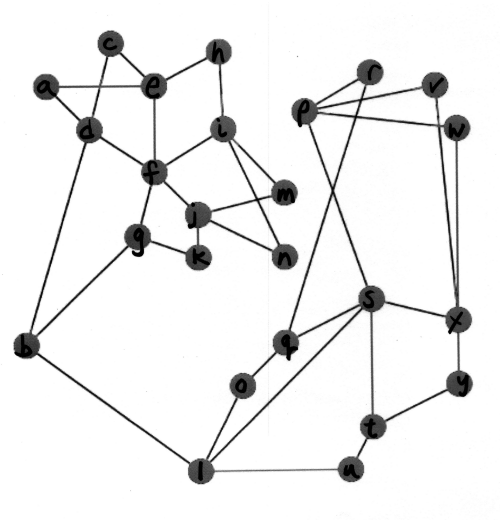


Figure A.18: Graph 13, a Greedy Resistant graph

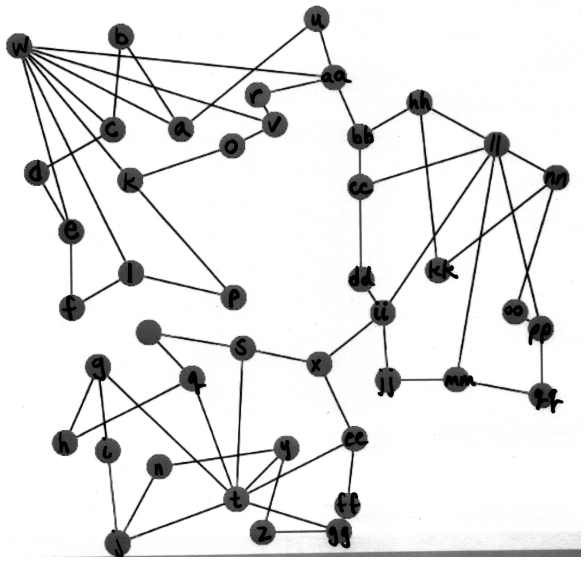


Figure A.19: Graph 14, a Greedy Resistant graph

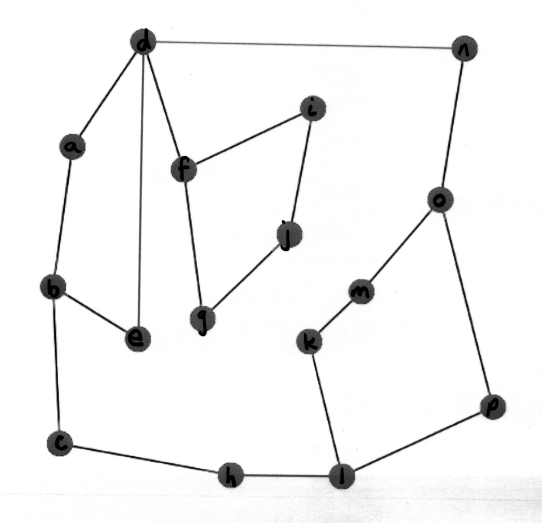


Figure A.20: Graph 15, a No Strategy graph

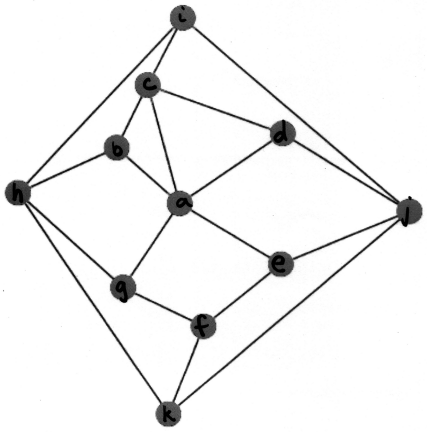


Figure A.21: Graph 16, a No Strategy graph

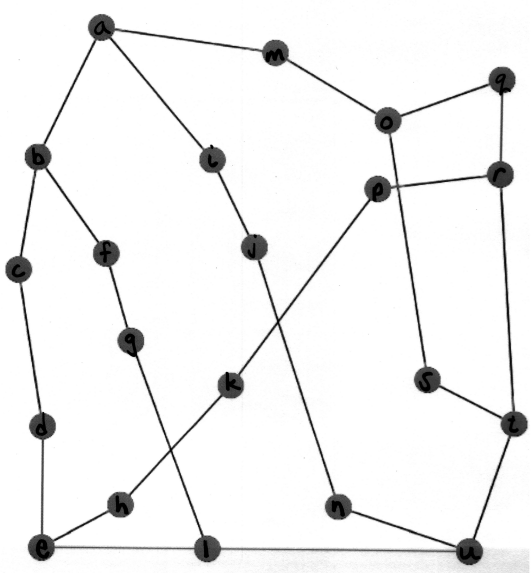


Figure A.22: Graph 17, a No Strategy graph

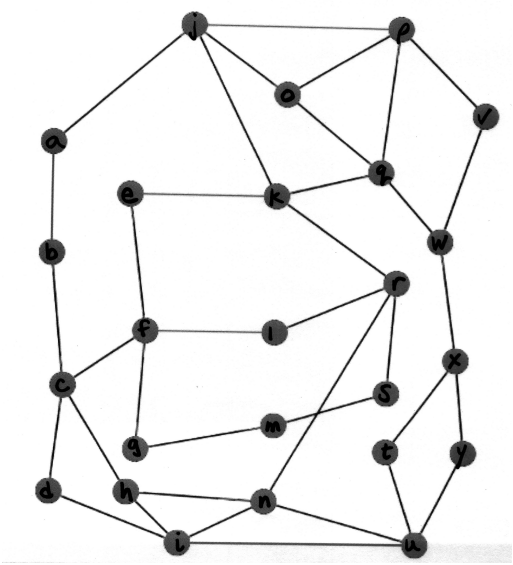


Figure A.23: Graph 18, a No Strategy graph

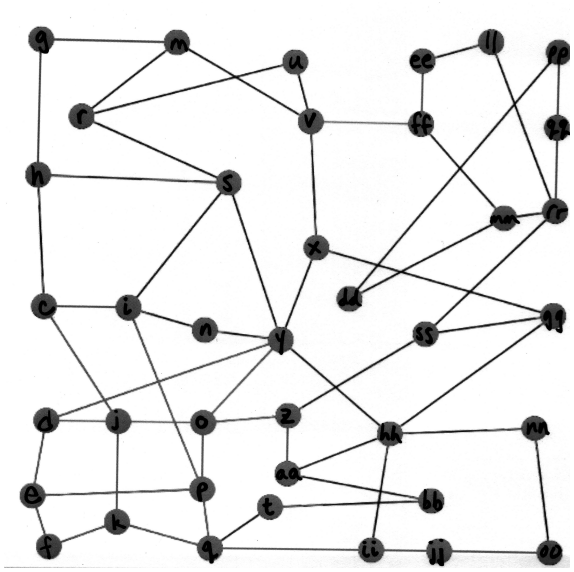


Figure A.24: Graph 19, a No Strategy graph

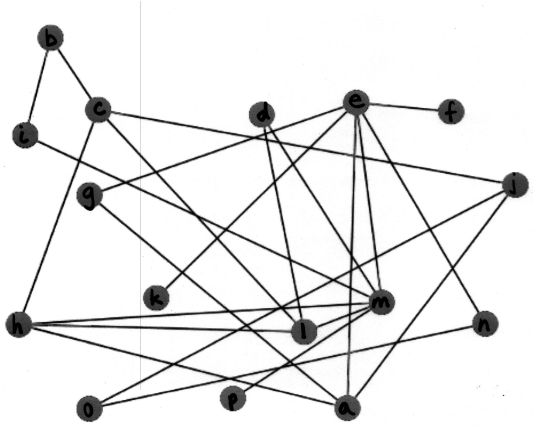


Figure A.25: Graph 20, a Randomly Generated graph

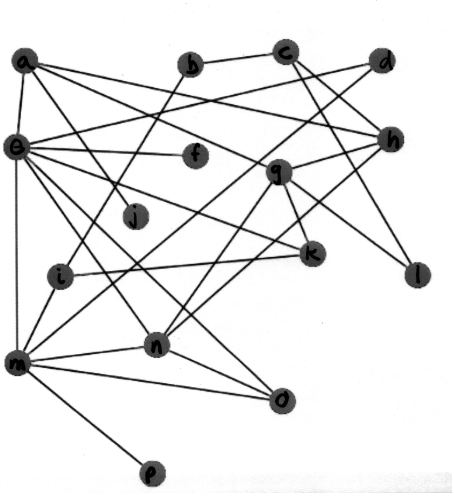


Figure A.26: Graph 21, a Randomly Generated graph

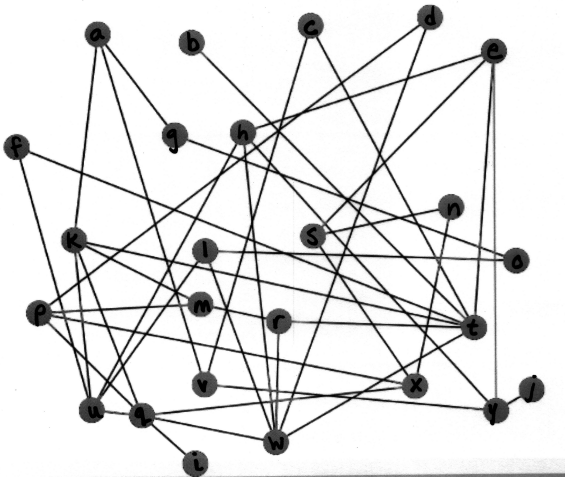


Figure A.27: Graph 22, a Randomly Generated graph

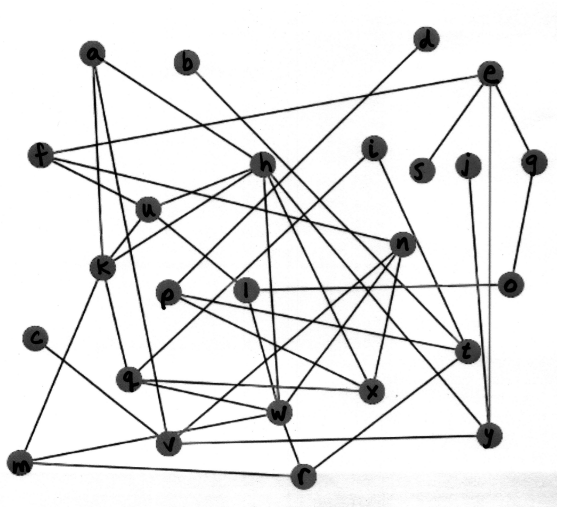


Figure A.28: Graph 23, a Randomly Generated graph

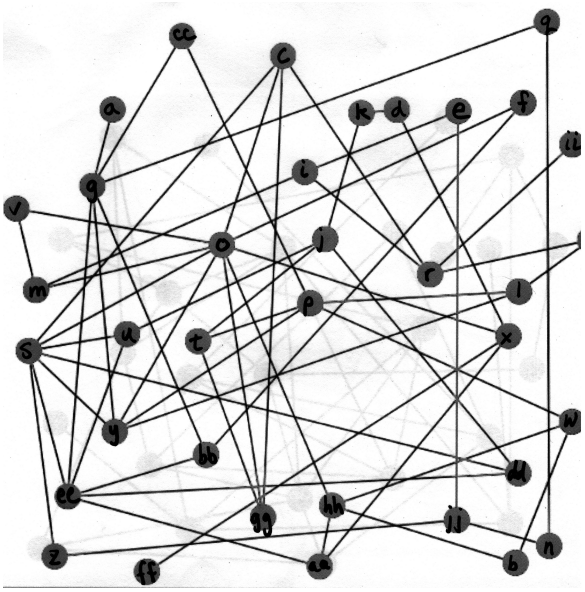


Figure A.29: Graph 24, a Randomly Generated graph

A.4 Results

A.4.1 Graphs

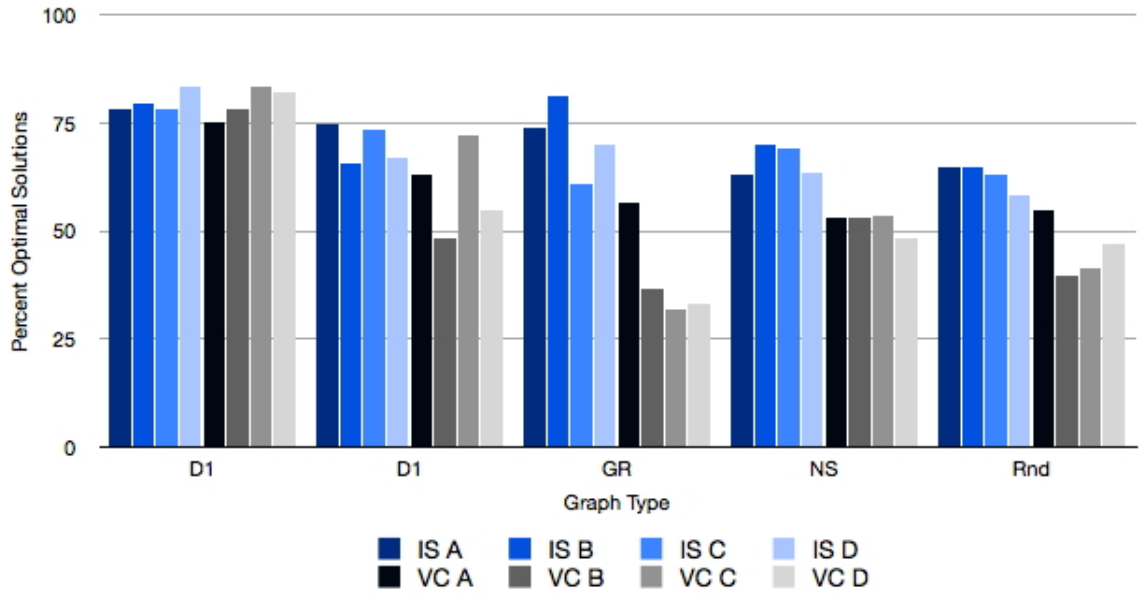


Figure A.30: Percent Optimal solutions found by Group

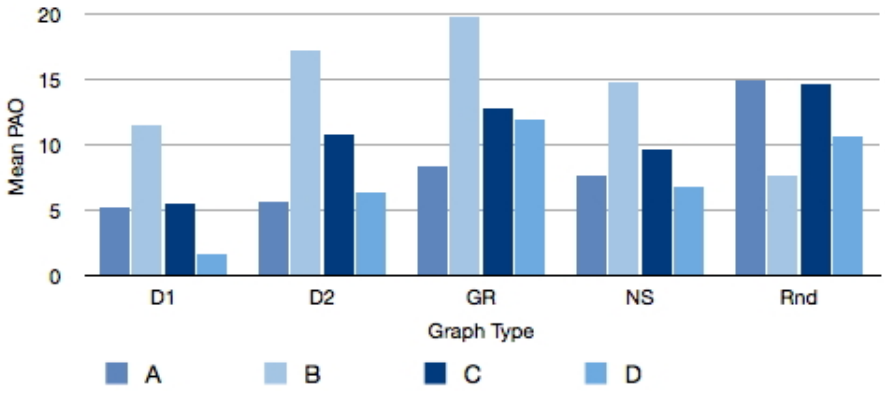


Figure A.31: Mean PAO by Group and Graph Type

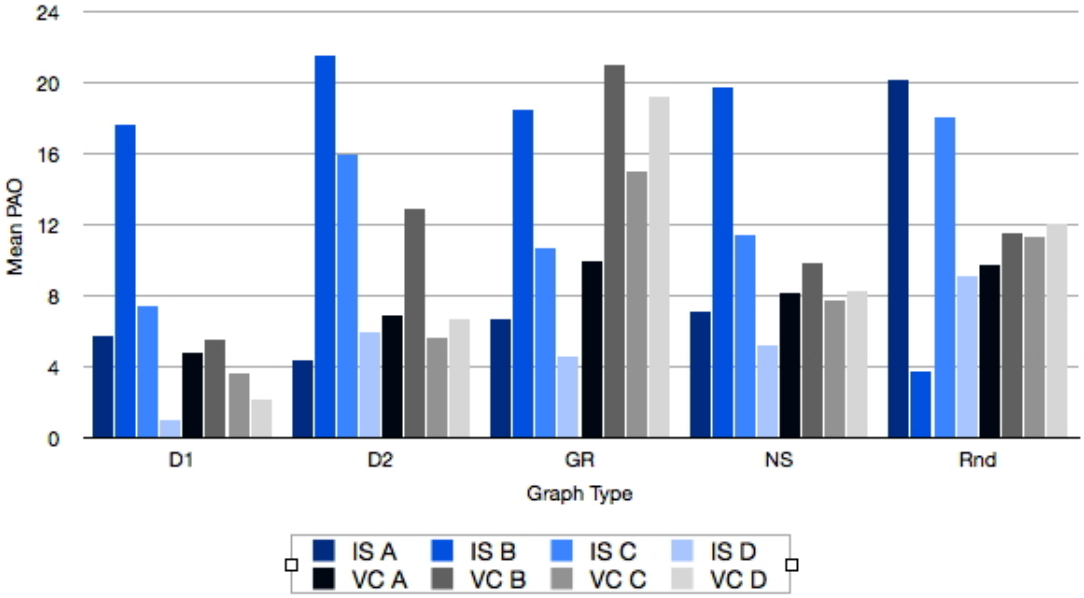


Figure A.32: Mean PAO by Group, Problem, and Graph Type

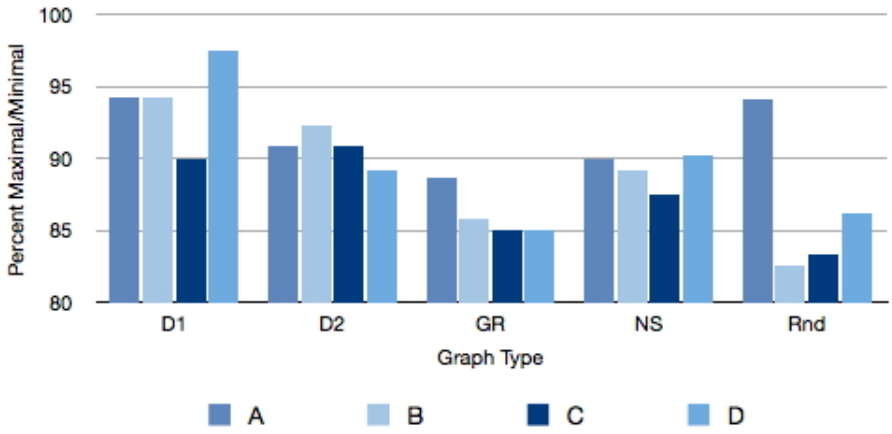


Figure A.33: Percent Maximal/Minimal solutions found by Group

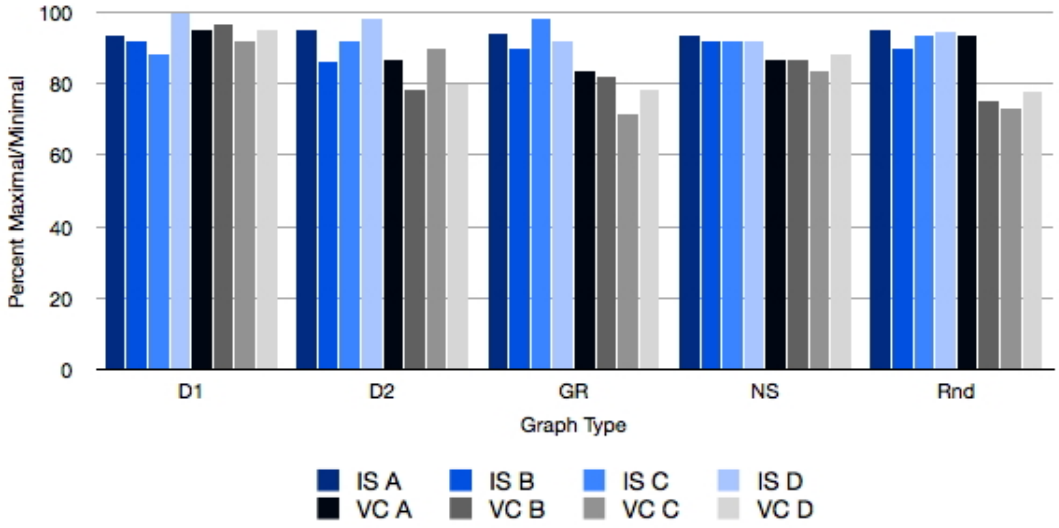


Figure A.34: Percent Maximal/Minimal solutions found by Problem, Group, and Graph Type

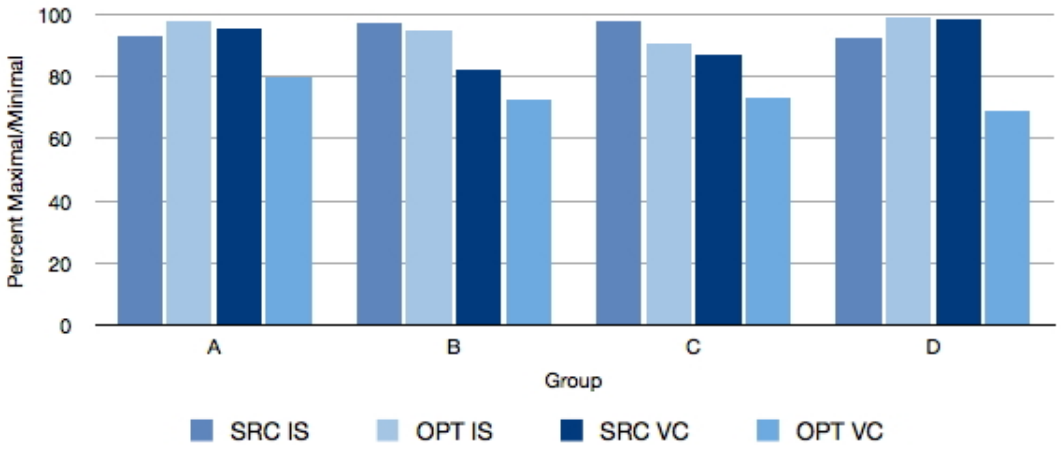


Figure A.35: Percent Maximal/Minimal solutions found by Problem, Problem Version and Group

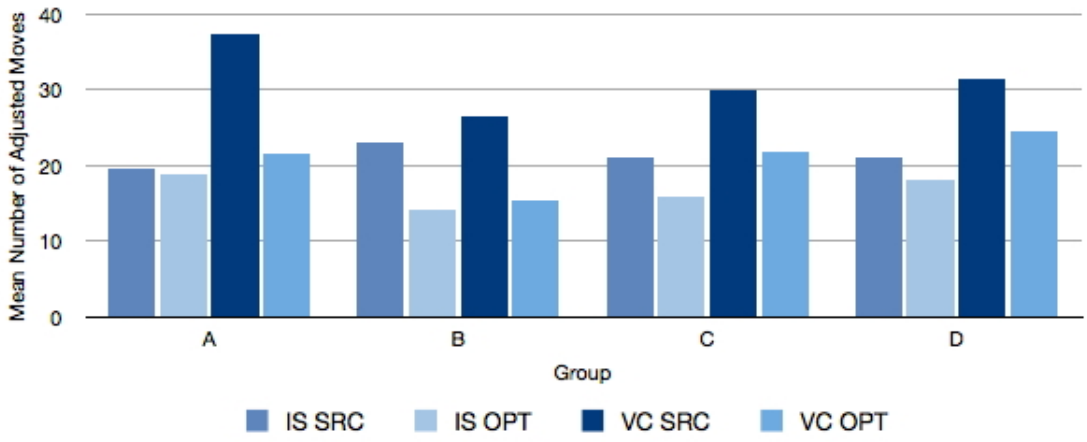


Figure A.36: Mean number of adjusted Moves by Group, Problem and Problem Version

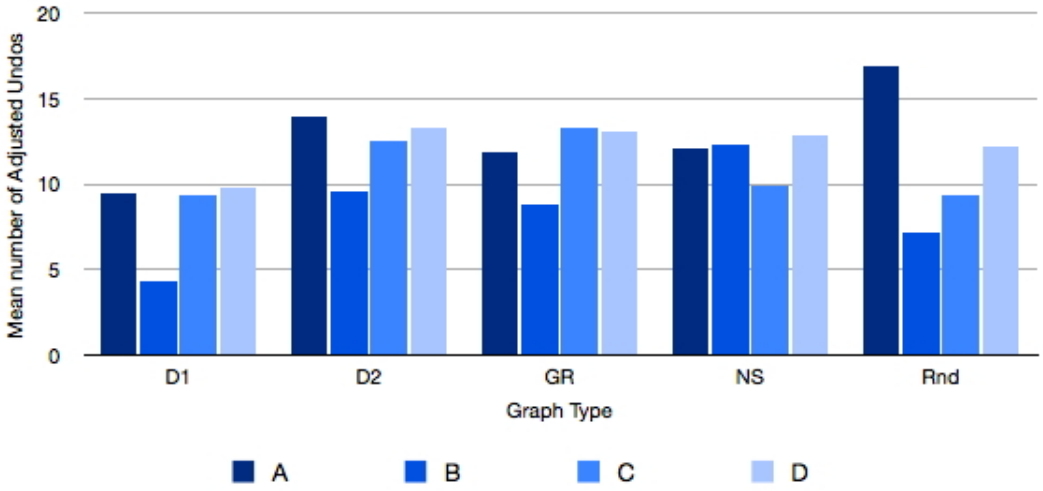


Figure A.37: Mean number of Undos by Group and Graph Type

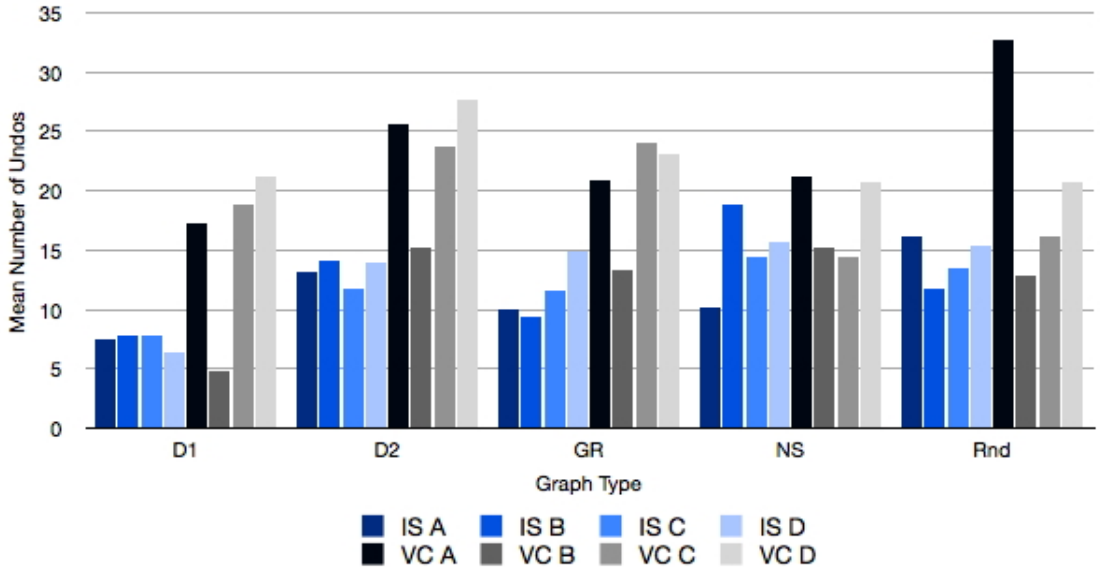


Figure A.38: Mean number of adjusted Undos by Problem, Group and Graph Type

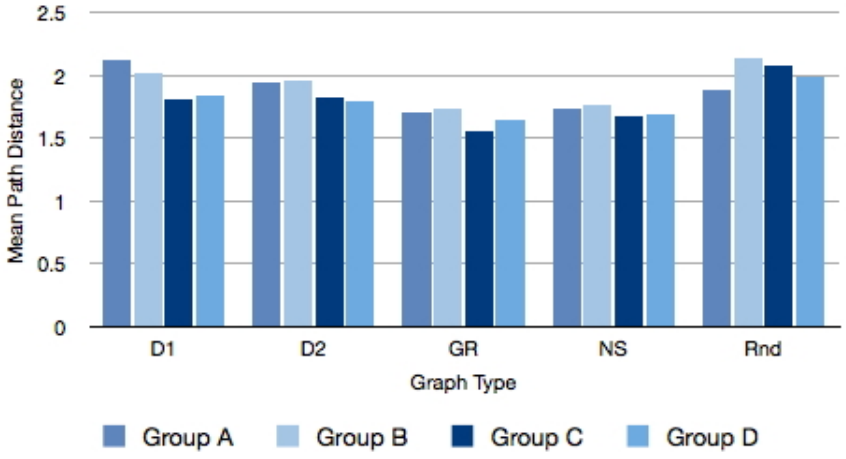


Figure A.39: Mean Path Distance between subsequent Moves, by Problem, Group and Graph Type

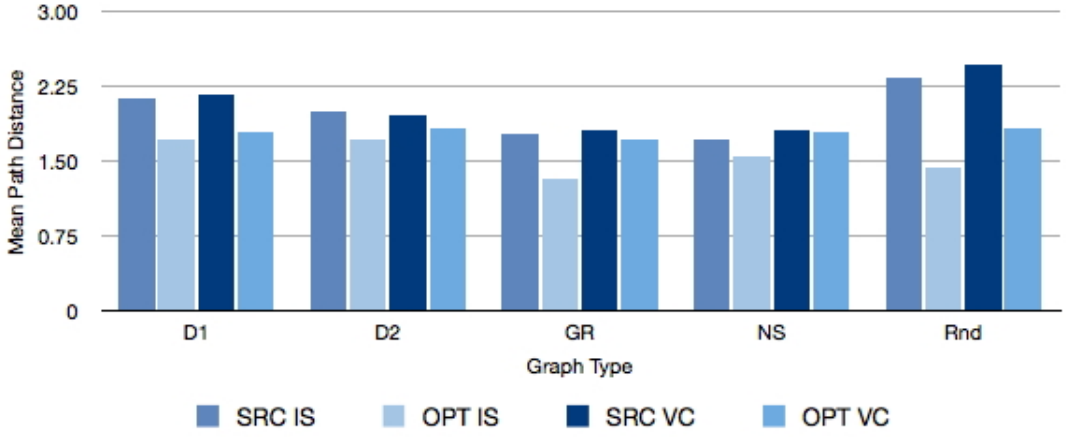


Figure A.40: Mean Path Distance between subsequent Moves, by Problem, Problem Version and Graph Type

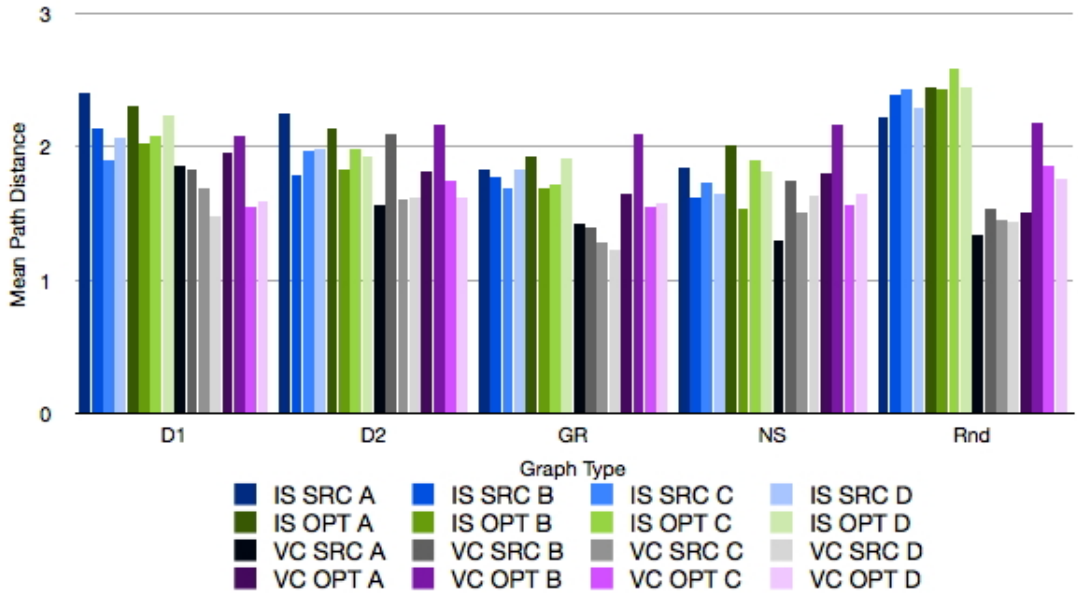


Figure A.41: Mean Path Distance between subsequent Moves all factors

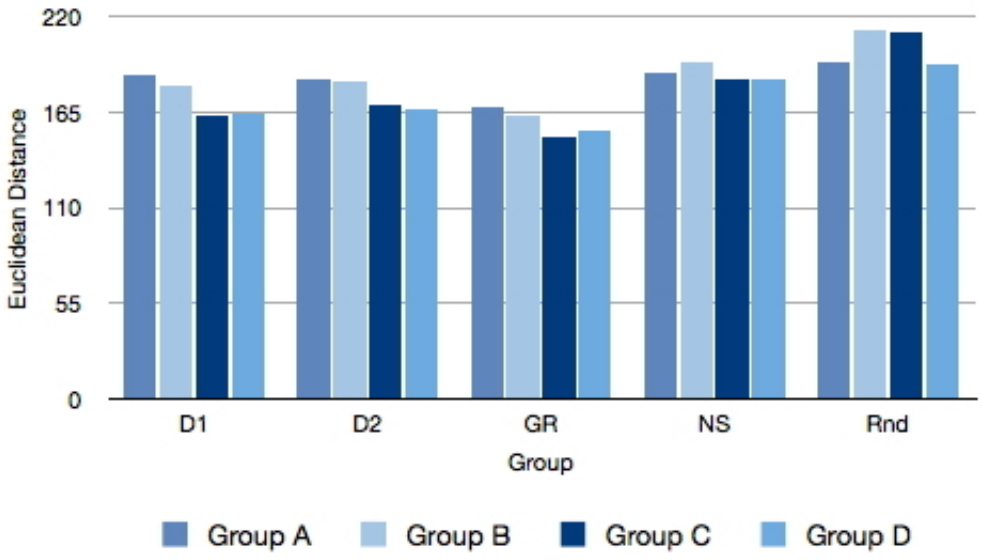


Figure A.42: Mean Euclidean Distance between subsequent Moves by Group and Graph Type

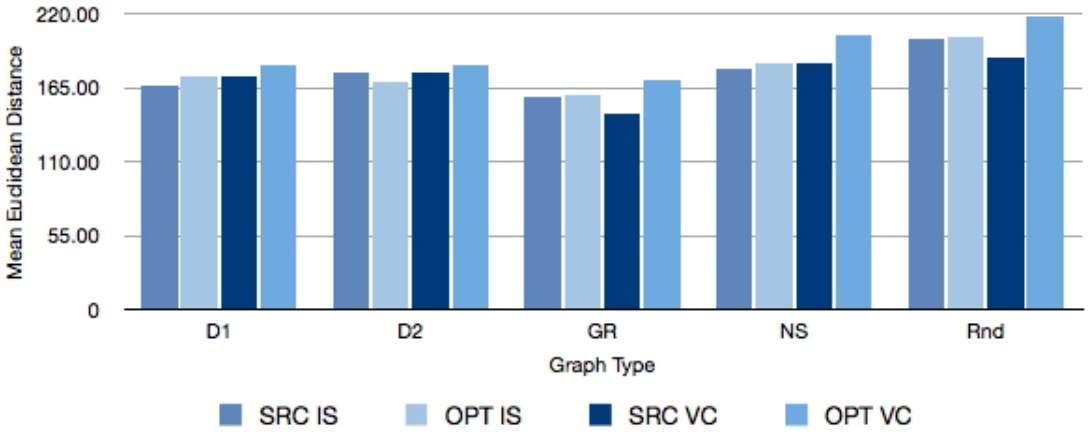


Figure A.43: Mean Euclidean Distance between subsequent Moves by Problem, Problem Version, and Graph Type

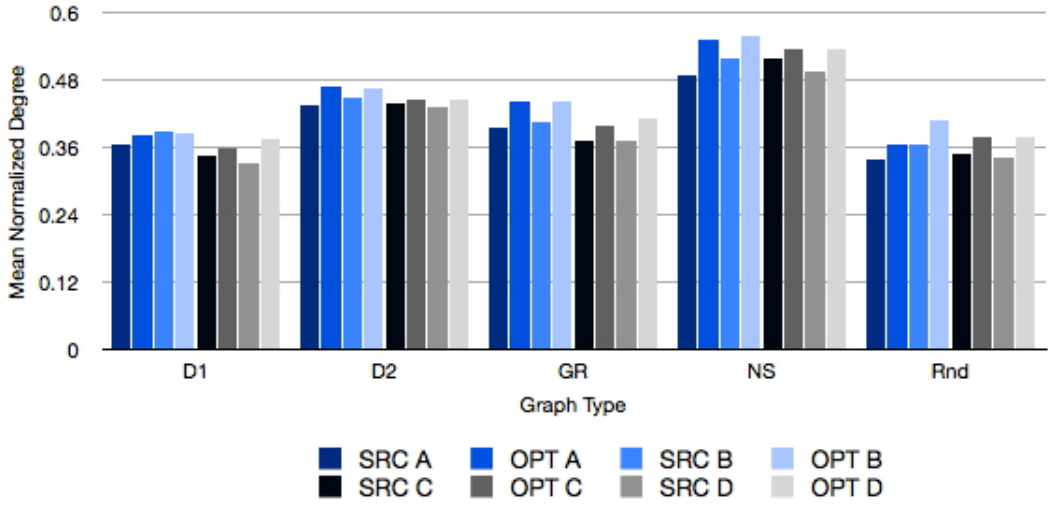


Figure A.44: Mean Degree of selections by Group and Graph Type

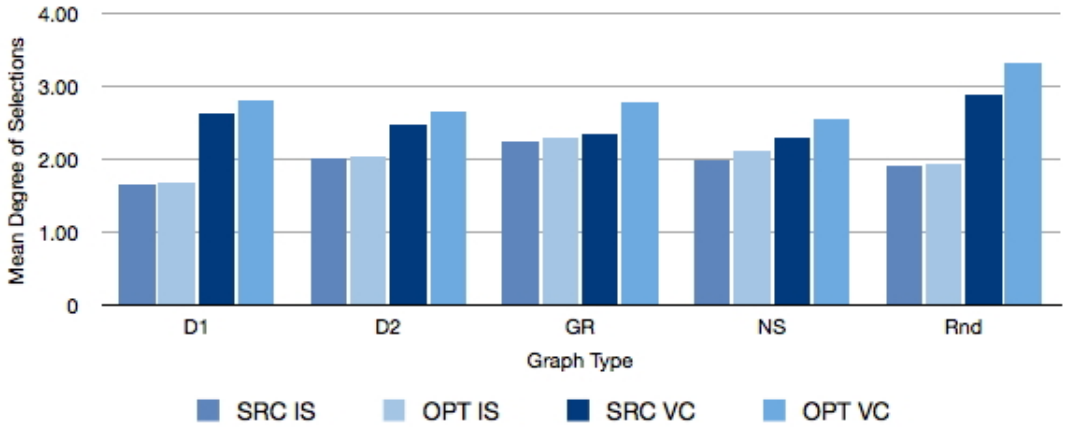


Figure A.45: Mean Degree of selections by Problem, Problem Version, and Graph Type

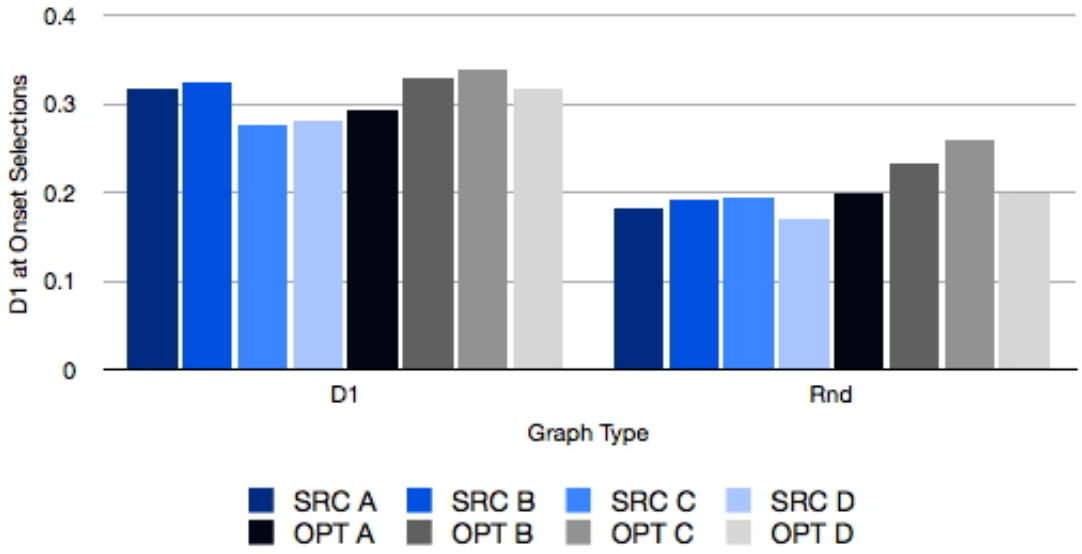


Figure A.46: Mean normalized D1 at Onset selections by Problem, Group and Graph Type

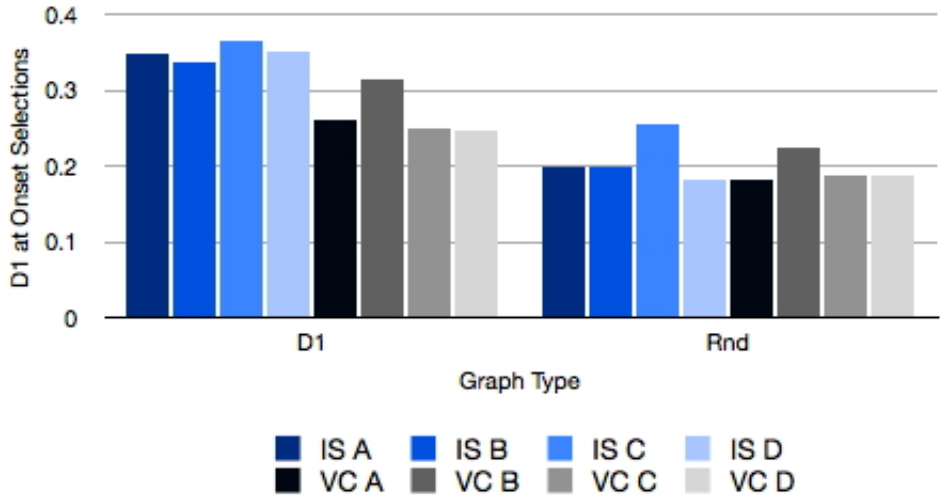


Figure A.47: Mean normalized D1 at Onset selections by Problem Version, Group and Graph Type

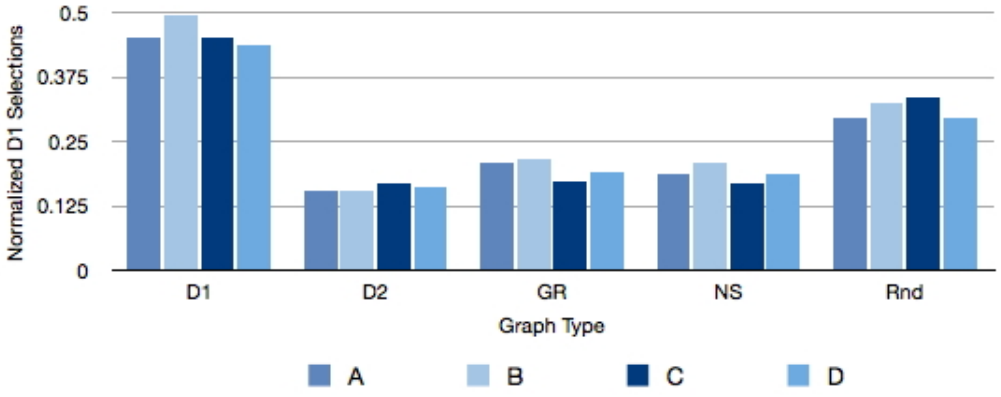


Figure A.48: Mean normalized D1 selections by Group and Graph Type

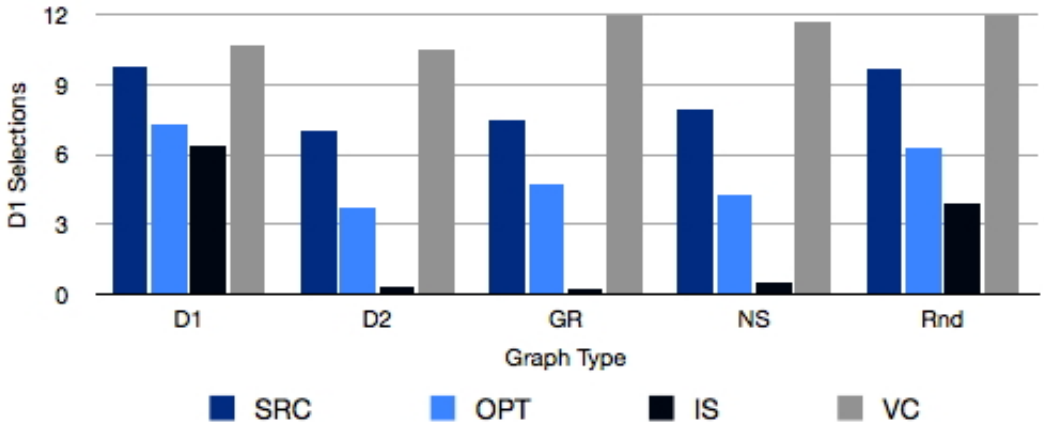


Figure A.49: Mean normalized D1 selections for both Problems and both Problem Versions

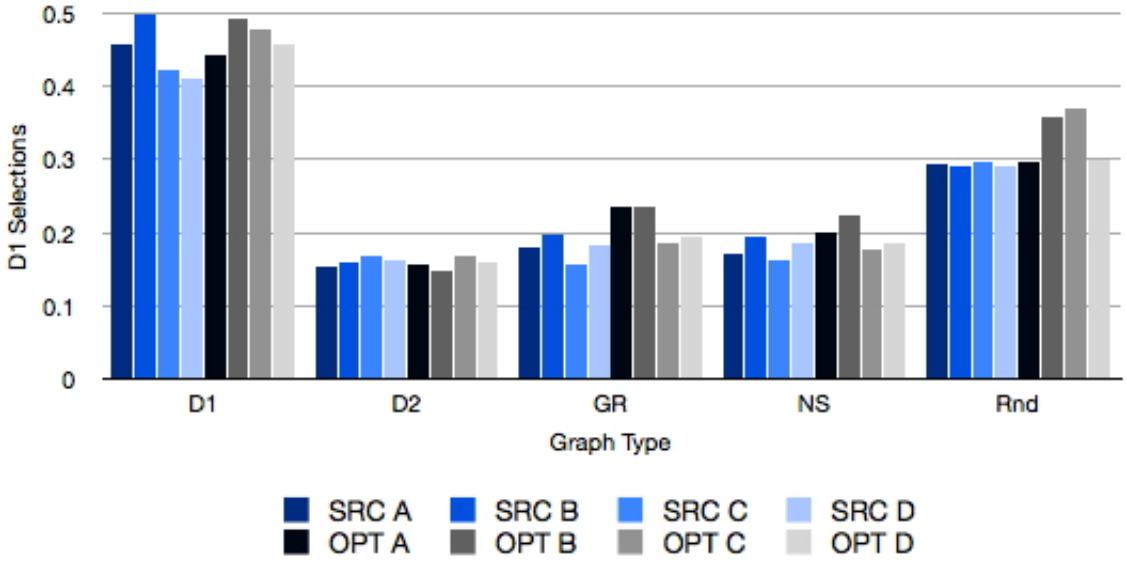


Figure A.50: Mean normalized D1 selections for Problem Version, Graph Type, and Group

A.4.2 Tables

Table A.2: Percent Optimal solutions.

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	Group A	76.6	69.0	65.2	58.3	59.9	65.81
	Group B	78.9	56.9	59.0	61.6	52.4	64.80
	Group C	80.8	72.6	46.7	61.2	52.4	55.75
	Group D	82.8	61.0	51.7	55.8	52.4	59.26
	All Groups	79.74	64.80	61.77	62.75	60.71	
Independent Set							
	Group A	78.2	74.8	73.6	63.2	64.8	0.71
	Group B	79.5	65.6	81.2	69.8	64.8	0.72
	Group C	78.1	73.3	61.0	68.9	63.1	0.69
	Group D	83.5	67.0	70.0	63.3	58.1	0.68
Vertex Cover							
	Group A	75.0	63.2	56.5	53.2	54.8	0.61
	Group B	78.3	48.2	36.5	53.2	39.8	0.51
	Group C	83.4	72.0	32.0	53.3	41.5	0.56
	Group D	82.1	54.9	33.2	48.2	47.0	0.53
Search							
	Group A	88.3	83.5	72.8	70.1	81.8	0.79
	Group B	94.6	73.5	78.7	80.1	75.2	0.80
	Group C	88.3	86.9	60.5	81.0	71.8	0.78
	Group D	78.2	76.7	61.9	63.1	68.5	0.70
	All Groups	88.0	80.0	69.0	74.0	74.0	
Optimization							
	Group A	65.0	54.9	57.9	46.7	38.4	0.53
	Group B	63.4	40.8	39.7	43.4	30.0	0.43
	Group C	73.4	58.5	33.3	41.7	33.4	0.48
	Group D	86.5	45.4	41.7	48.3	36.6	0.52
	All Groups	72.0	50.0	43.0	45.0	35.0	

Table A.3: Mean PAO

		D1	**D2**	**GR**	**NS**	**Rnd**
All Conditions						
	Group A	5.251	5.634	8.326	7.628	14.976
	Group B	11.570	17.221	19.751	14.783	7.594
	Group C	5.556	10.820	12.827	9.581	14.684
	Group D	1.616	6.289	11.893	6.760	10.575
	All Groups	5.998	9.991	13.199	9.688	11.957
Independent Set						
	Group A	5.718	4.397	6.718	7.120	20.185
	Group B	17.626	21.558	18.517	19.731	3.704
	Group C	7.439	15.984	10.644	11.426	18.056
	Group D	1.025	5.929	4.610	5.241	9.105
	All Groups	7.952	11.967	10.112	10.880	12.762
Vertex Cover						
	Group A	4.784	6.872	9.934	8.136	9.766
	Group B	5.514	12.884	20.985	9.835	11.484
	Group C	3.673	5.655	15.010	7.737	11.313
	Group D	2.207	6.649	19.177	8.279	12.045
	All Groups	4.045	8.015	16.277	8.497	11.152

Table A.4: Percent Maximal/Minimal solutions.

		D1	**D2**	**GR**	**NS**	**Rnd**
All Conditions						
	Group A	94.2	90.8	88.7	90.0	94.1
	Group B	94.2	92.3	85.8	89.2	82.5
	Group C	90.0	90.8	85.0	87.5	83.3
	Group D	97.5	89.1	85.0	90.2	86.2
Independent Set						
	Group A	93.3	95.0	94.2	93.3	95.0
	Group B	91.7	86.2	90.0	91.7	90.0
	Group C	88.3	91.7	98.3	91.7	93.3
	Group D	100.0	98.3	91.7	92.1	94.7
Vertex Cover						
	Group A	95.0	86.7	83.3	86.7	93.2
	Group B	96.7	78.3	81.7	86.7	74.9
	Group C	91.7	90.0	71.7	83.3	73.2
	Group D	94.9	79.9	78.2	88.2	77.6
Search						
	Group A	95.0	96.7	94.6	91.7	100.0
	Group B	100.0	95.0	95.0	96.7	93.3
	Group C	95.0	96.7	91.7	95.0	93.3
	Group D	100.0	96.6	91.6	93.7	96.2
Optimization						
	Group A	93.3	85.0	82.9	88.3	88.3
	Group B	88.3	69.6	76.7	81.7	71.7
	Group C	85.0	85.0	78.3	80.0	73.3
	Group D	95.0	81.7	78.4	86.7	76.5

Table A.5: Mean Time (s) per Move.

		D1	**D2**	**GR**	**NS**	**Rnd**
All Conditions						
	Group A	2.623	2.380	1.998	2.242	2.254
	Group B	2.938	3.185	2.928	3.324	2.767
	Group C	1.950	2.339	2.093	2.091	2.562
	Group D	1.845	2.000	2.117	3.038	2.194
	All Groups	2.339	2.476	2.284	2.674	2.447
Independent Set						
	Group A	2.344	2.511	1.716	2.084	2.448
	Group B	3.287	2.860	3.532	3.628	2.546
	Group C	1.886	2.840	1.955	2.266	2.905
	Group D	1.829	2.147	2.197	3.272	2.380
	All Groups	2.34	2.59	2.35	2.81	2.57
Vertex Cover						
	Group A	2.901	2.248	2.280	2.400	2.058
	Group B	2.589	3.511	2.325	3.020	2.986
	Group C	2.014	1.838	2.231	1.915	2.217
	Group D	1.859	1.851	2.036	2.802	2.003
	Al Groups	2.34	2.36	2.22	2.53	2.32
Search						
	Group A	2.458	2.218	1.872	2.052	2.394
	Group B	2.300	3.147	2.070	2.801	2.527
	Group C	1.863	1.996	1.980	1.978	2.555
	Group D	1.904	2.041	2.078	2.080	2.379
	All Groups	2.13	2.35	2.00	2.48	2.47
Optimization						
	Group A	2.785	2.540	2.122	2.430	2.120
	Group B	3.574	3.221	3.784	3.844	3.012
	Group C	2.034	2.679	2.204	2.201	2.575
	Group D	1.790	1.963	2.160	2.999	2.017
	All Groups	2.55	2.60	2.57	2.87	2.43

Table A.6: Mean number of Toggles

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	Group A	3.079	5.837	5.354	5.572	8.229	5.162
	Group B	2.099	5.895	5.145	5.747	6.037	4.982
	Group C	3.979	7.5	7.020	6.564	6.429	6.296
	Group D	4.029	8.115	7.632	6.843	7.253	6.778
	All Groups	3.296	6.837	6.288	6.159	7.007	5.015
Independent Set							
	Group A	2.423	4.223	5.673	3.156	6.357	4.37
	Group B	2.846	6.989	4.873	6.406	6.191	5.46
	Group C	2.573	4.956	4.789	5.756	6.241	4.86
	Group D	2.623	5.890	6.106	5.490	7.030	5.43
	All Groups	2.62	5.51	5.36	5.24	6.40	
Vertex Cover							
	Group A	3.739	7.455	5.039	8.005	10.089	6.86
	Group B	1.355	4.805	5.422	5.105	5.873	4.51
	Group C	5.389	10.047	9.255	7.389	6.606	7.73
	Group D	5.439	10.343	9.160	7.977	7.668	8.14
	All Groups	3.98	8.16	7.22	7.11	7.57	
Search							
	Group A	3.965	7.345	7.196	8.063	9.710	7.20
	Group B	3.236	7.796	8.996	9.063	10.660	7.94
	Group C	5.280	10.705	8.096	9.496	8.427	8.40
	Group D	4.977	9.098	9.248	7.915	9.060	8.06
Optimization							
	Group A	2.471	4.338	3.521	3.104	6.735	4.03
	Group B	0.971	4.004	1.304	2.454	1.401	2.03
	Group C	2.688	4.304	5.954	3.654	4.418	4.20
	Group D	3.085	7.135	6.018	5.551	5.686	5.49

Table A.7: Adjusted Moves to Find a Solution

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	Group A	23.72	31.15	27.69	25.73	36.02	24.28
	Group B	17.37	26.56	24.33	27.46	23.43	19.74
	Group C	24.15	29.40	29.67	24.80	26.25	22.16
	Group D	25.36	32.77	30.96	28.64	29.56	27.96
	All Groups	19.46	24.23	23.80	22.11	22.80	
Independent Set							
	Group A	18.46	19.46	19.90	17.15	21.36	19.72
	Group B	17.43	18.06	18.20	22.67	16.76	18.63
	Group C	17.38	17.30	20.06	19.39	18.34	18.51
	Group D	17.00	18.51	21.87	20.44	19.93	19.55
	All Groups	17.57	18.34	20.00	19.92	10.11	
Vertex Cover							
	Group A	23.39	32.17	28.42	27.13	35.59	29.27
	Group B	13.41	25.06	23.56	22.70	19.87	20.90
	Group C	23.06	31.33	30.61	21.37	23.28	25.86
	Group D	25.70	32.15	28.31	26.20	27.46	27.96
	All Groups	21.38	30.17	27.66	24.34	26.54	
Search							
	Group A	24.79	30.06	26.21	24.43	36.43	28.43
	Group B	18.0	24.83	27.73	30.01	23.73	24.85
	Group C	24.44	29.55	27.61	21.74	24.19	25.48
	Group D	26.37	27.07	25.62	26.02	25.83	26.19
	All Groups	23.39	27.87	26.80	25.55	27.59	
Optimization							
	Group A	17.05	21.55	22	19.84	20.48	20.19
	Group B	12.87	18.31	13.97	15.38	12.90	14.68
	Group C	15.99	19.08	22.85	19.01	17.49	18.89
	Group D	16.34	23.59	24.48	20.62	21.49	21.29
	All Groups	15.57	20.64	20.81	18.72	18.08	

Table A.8: Mean number of adjusted Undos

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	Group A	9.48	13.94	11.89	12.07	16.83	12.85
	Group B	4.37	9.56	8.78	12.26	7.11	8.41
	Group C	9.30	12.55	13.28	9.94	9.36	10.87
	Group D	9.74	13.32	13.02	12.79	12.22	12.22
	All Groups	8.221	12.34	11.72	11.77	13.47	
Independent Set							
	Group A	7.45	13.11	10.06	10.22	16.19	7.86
	Group B	7.83	14.07	9.32	18.87	11.72	7.86
	Group C	7.74	11.82	11.65	14.39	13.52	7.46
	Group D	6.43	13.89	14.89	15.64	15.39	8.21
	All Groups	4.95	8.17	7.60	10.39	8.12	
Vertex Cover							
	Group A	17.25	25.60	20.96	21.25	32.75	17.91
	Group B	4.86	15.19	13.35	15.23	12.82	13.53
	Group C	18.82	23.68	24.04	14.41	16.09	14.46
	Group D	21.18	27.74	23.13	20.73	20.77	15.26
	All Groups	11.54	16.56	15.92	13.18	14.66	
Search							
	Group A	16.50	25.35	17.87	19.77	33.63	17.48
	Group B	9.47	19.15	21.69	27.08	21.47	13.53
	Group C	18.99	23.86	18.63	17.07	19.19	14.46
	Group D	19.76	23.42	20.41	21.66	21.50	15.26
	All Groups	12.07	16.27	15.26	15.92	16.38	
Optimization							
	Group A	8.19	13.36	12.93	11.69	15.30	8.29
	Group B	3.30	10.19	2.81	7.10	3.14	3.35
	Group C	7.53	11.59	16.23	11.69	10.36	7.32
	Group D	7.91	18.27	17.43	14.77	14.72	9.21
	All Groups	4.43	8.47	8.22	7.6	6.44	

Table A.9: Mean Path Length between subsequent selections

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	Group A	2.127	1.941	1.705	1.739	1.879	1.878
	Group B	2.020	1.964	1.739	1.765	2.131	1.923
	Group C	1.801	1.822	1.559	1.673	2.080	1.787
	Group D	1.842	1.791	1.639	1.685	1.985	1.788
	All Groups	1.947	1.88	1.66	1.716	2.019	1.844
Independent Set							
	Group A	2.351	2.193	1.875	1.931	2.331	2.14
	Group B	2.081	1.809	1.735	1.575	3.404	1.92
	Group C	1.990	1.972	1.705	1.813	2.505	2.00
	Group D	2.147	1.957	1.870	1.727	2.366	2.01
	All Groups	2.14	1.98	1.80	1.76	2.40	2.017
Vertex Cover							
	Group A	1.903	1.690	1.536	1.548	1.422	1.62
	Group B	1.959	2.120	1.744	1.954	1.854	1.93
	Group C	1.613	1.673	1.414	1.534	1.649	1.58
	Group D	1.532	1.619	1.403	1.639	1.598	1.56
	All Groups	1.75	1.78	1.52	1.67	1.63	1.671
Search							
	Group A	2.124	1.907	1.628	1.570	1.785	1.80
	Group B	2.130	1.975	1.782	1.908	1.972	1.83
	Group C	1.987	1.939	1.582	1.684	1.961	1.72
	Group D	2.052	1.99	1.895	1.845	2.301	1.72
	All Groups	1.92	1.86	1.56	1.63	1.89	1.769
Optimization							
	Group A	1.789	1.783	1.485	1.617	1.937	1.95
	Group B	1.813	1.961	1.633	1.729	2.222	2.02
	Group C	1.771	1.804	1.526	1.644	1.8681	1.85
	Group D	1.911	1.776	1.751	1.726	2.1	1.85
	All Groups	1.98	1.90	1.77	1.80	2.15	1.919

Table A.10: Mean Euclidean Distance between subsequent selections

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	Group A	186.14	183.21	167.54	187.28	193.60	183.5
	Group B	180.42	182.65	162.68	193.46	212.32	186.3
	Group C	163.43	169.36	150.27	183.95	210.99	175.6
	Group D	163.82	167.11	153.82	184.20	192.44	172.3
	All Groups	173.5	175.6	158.6	187.2	202.4	179.41
Independent Set							
	Group A	186.43	193.03	170.04	192.23	204.32	189.21
	Group B	162.91	162.74	154.46	167.43	197.5	169.00
	Group C	160.69	168.60	153.27	183.67	214.73	176.19
	Group D	168.52	167.14	157.23	180.57	192.62	173.23
	All Groups	169.62	172.86	158.74	180.96	202.35	176.9
Vertex Cover							
	Group A	185.89	173.44	165.08	182.36	182.79	177.82
	Group B	197.97	202.59	170.94	219.52	227.04	203.52
	Group C	166.21	170.15	147.31	184.27	207.17	174.93
	Group D	159.06	167.02	150.35	187.75	192.37	171.23
	All Groups	177.3	178.32	158.43	193.49	202.49	181.9
Search							
	Group A	186.29	180.97	164.13	179.07	191.66	180.34
	Group B	174.81	180.41	151.71	184.70	201.52	178.54
	Group C	160.94	171.70	146.58	183.40	202.13	172.86
	Group D	156.16	169.70	146.04	176.40	181.48	165.97
	All Groups	169.57	175.71	152.13	180.91	194.40	174.5
Optimization							
	Group A	186.00	185.46	170.95	195.48	195.45	186.66
	Group B	186.03	184.89	173.65	202.21	223.01	193.95
	Group C	165.92	167.01	153.96	184.50	219.76	178.23
	Group D	171.40	164.44	161.53	191.92	203.12	178.50
	All Groups	177.32	175.44	165.01	193.51	210.40	184.3

Table A.11: Mean normalized number of D1 selections

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	All Groups	0.4571	0.1593	0.1959	0.1877	0.3127	
Independent Set							
	Group A	0.36	0.01	0.01	0.01	0.20	0.12
	Group B	0.34	0.01	0.00	0.01	0.20	0.11
	Group C	0.37	0.01	0.01	0.01	0.26	0.13
	Group D	0.35	0.01	0.01	0.02	0.18	0.11
	All Groups	0.35	0.01	0.01	0.01	0.21	
Vertex Cover							
	Group A	0.55	0.30	0.41	0.36	0.39	0.40
	Group B	0.65	0.30	0.43	0.40	0.45	0.45
	Group C	0.53	0.33	0.34	0.33	0.41	0.39
	Group D	0.52	0.32	0.37	0.35	0.41	0.39
	All Groups	0.56	0.31	0.39	0.36	0.41	
Search							
	Group A	0.46	0.15	0.18	0.17	0.30	0.25
	Group B	0.50	0.16	0.20	0.19	0.29	0.27
	Group C	0.42	0.17	0.16	0.16	0.30	0.24
	Group D	0.41	0.16	0.18	0.19	0.29	0.25
	All Groups	0.45	0.16	0.18	0.18	0.29	
Optimization							
	Group A	0.44	0.16	0.23	0.20	0.30	0.27
	Group B	0.49	0.15	0.24	0.22	0.36	0.29
	Group C	0.48	0.17	0.19	0.18	0.37	0.28
	Group D	0.46	0.16	0.20	0.19	0.30	0.26
	All Groups	0.47	0.16	0.21	0.20	0.33	

Table A.12: Mean normalized number of D1 at Onset selections

		D1	**Rnd**	**All**
All Conditions				
	Group A	0.305	0.191	0.2479
	Group B	0.326	0.212	0.2694
	Group C	0.307	0.227	0.2674
	Group D	0.299	0.184	0.2416
	All groups	0.3091	0.2037	
Independent Set				
	Group A	0.348	0.199	0.274
	Group B	0.338	0.200	0.269
	Group C	0.365	0.255	0.310
	Group D	0.352	0.183	0.267
	All Groups	0.35	0.21	
Vertex Cover				
	Group A	0.261	0.182	0.222
	Group B	0.314	0.225	0.270
	Group C	0.249	0.186	0.224
	Group D	0.246	0.186	0.216
	All Groups	0.27	0.20	
Search				
	Group A	0.317	0.182	0.250
	Group B	0.323	0.192	0.258
	Group C	0.276	0.195	0.236
	Group D	0.280	0.169	0.225
	All Groups	0.30	0.18	
Optimization				
	Group A	0.292	0.199	0.245
	Group B	0.329	0.232	0.281
	Group C	0.338	0.259	0.299
	Group D	0.317	0.199	0.258
	All Groups	0.32	0.22	

Table A.13: Mean normalized number of D2 selections

		D2	**Rnd**	**All**
All Conditions				
	Group A	0.135	0.046	0.09056
	Group B	0.131	0.053	0.09236
	Group C	0.139	0.044	0.09166
	Group D	0.133	0.044	0.08876
	All Groups	0.1346	0.0466	0.0908
Independent Set				
	Group A	0.231	0.070	0.151
	Group B	0.209	0.071	0.140
	Group C	0.236	0.065	0.151
	Group D	0.233	0.065	0.148
	All Groups	0.23	0.07	
Vertex Cover				
	Group A	0.038	0.021	0.030
	Group B	0.052	0.035	0.044
	Group C	0.041	0.022	0.032
	Group D	0.034	0.024	0.029
	All Groups	0.04	0.03	
Search				
	Group A	0.142	0.047	0.095
	Group B	0.134	0.048	0.092
	Group C	0.135	0.049	0.092
	Group D	0.143	0.042	0.093
	All Groups	0.14	0.05	
Optimization				
	Group A	0.128	0.044	0.086
	Group B	0.129	0.057	0.093
	Group C	0.143	0.039	0.091
	Group D	0.123	0.045	0.084
	All Groups	0.13	0.05	

Table A.14: Mean normalized number of D2 at Onset selections

		D2	**Rnd**	**All**
All Conditions				
	Group A	0.241	0.108	0.175
	Group B	0.242	0.126	0.184
	Group C	0.250	0.116	0.184
	Group D	0.249	0.103	0.176
	All groups	0.246	0.113	0.180
Independent Set				
	Group A	0.238	0.074	0.156
	Group B	0.216	0.072	0.144
	Group C	0.235	0.067	0.151
	Group D	0.240	0.069	0.155
	All Groups	0.23	0.07	
Vertex Cover				
	Group A	0.245	0.143	0.195
	Group B	0.268	0.180	0.225
	Group C	0.266	0.166	0.216
	Group D	0.259	0.137	0.198
	All Groups	0.26	0.16	
Search				
	Group A	0.251	0.11	0.181
	Group B	0.248	0.118	0.184
	Group C	0.247	0.114	0.181
	Group D	0.253	0.094	0.175
	All Groups	0.25	0.11	
Optimization				
	Group A	0.231	0.107	0.169
	Group B	0.236	0.133	0.184
	Group C	0.254	0.118	0.186
	Group D	0.245	0.111	0.178
	All Groups	0.24	0.12	

Table A.15: Mean Degree of selections

		D1	**D2**	**GR**	**NS**	**Rnd**	**All**
All Conditions							
	Group A	2.230	2.320	2.486	2.220	2.412	2.333
	Group B	2.308	2.362	2.533	2.298	2.644	2.429
	Group C	2.101	2.270	2.314	2.241	2.492	2.283
	Group D	2.107	2.259	2.359	2.186	2.485	2.279
	All Groups	2.186	2.303	2.423	2.236	2.508	
Independent Set							
	Group A	1.762	2.131	2.302	2.078	1.968	2.05
	Group B	1.581	1.989	2.255	1.980	1.848	1.93
	Group C	1.602	2.011	2.276	2.125	1.903	1.98
	Group D	1.707	2.007	0.264	2.028	1.986	2.00
	All Groups	1.66	2.03	2.27	2.05	1.93	
Vertex Cover							
	Group A	2.698	2.509	2.671	2.363	2.861	2.62
	Group B	3.035	2.735	2.812	2.617	3.444	2.93
	Group C	2.601	2.530	2.352	2.358	3.085	2.58
	Group D	2.510	2.515	2.458	2.347	2.989	2.56
	All Groups	2.71	2.57	2.57	2.42	3.10	
Search							
	Group A	2.180	2.231	2.343	2.086	2.317	2.23
	Group B	2.328	2.301	2.419	2.206	2.492	2.35
	Group C	2.059	2.241	2.230	2.201	2.380	2.22
	Group D	1.976	2.232	2.221	2.084	2.360	2.17
Optimization							
	Group A	2.279	2.408	2.629	2.355	2.507	2.44
	Group B	2.287	2.423	2.648	2.390	2.794	2.51
	Group C	2.142	2.299	2.397	2.280	2.602	2.34
	Group D	2.237	2.286	2.497	2.286	2.609	2.38

Table A.16: Mean Path Length between subsequent selections, all factors

Conditions and Group	**D1**	**D2**	**GR**	**NS**	**Rnd**
IS SRC A	2.397	2.251	1.830	1.846	2.221
IS SRC B	2.141	1.787	1.777	1.617	2.382
IS SRC C	1.894	1.967	1.689	1.731	2.424
IS SRC D	2.064	1.988	1.821	1.646	2.287
IS OPT A	2.306	2.135	1.921	2.016	2.440
IS OPT B	2.021	1.831	1.693	1.535	2.425
IS OPT C	2.086	1.978	1.722	1.895	2.586
IS OPT D	2.228	1.926	1.916	1.807	2.444
VC SRC A	1.850	1.561	1.425	1.293	1.341
VC SRC B	1.832	2.090	1.387	1.750	1.533
VC SRC C	1.683	1.598	1.281	1.502	1.443
VC SRC D	1.473	1.616	1.225	1.636	1.441
VC OPT A	1.956	1.818	1.646	1.803	1.499
VC OPT B	2.085	2.159	2.100	2.158	2.171
VC OPT C	1.542	1.746	1.546	1.565	1.852
VC OPT D	1.590	1.622	1.580	1.640	1.752

Table A.17: Mean Euclidean Distance between subsequent selections, all factors

Conditions and Group	**D1**	**D2**	**GR**	**NS**	**Rnd**
IS SRC A	190.56	197.15	172.62	188.56	206.85
IS SRC B	164.89	159.81	157.69	169.00	198.75
IS SRC C	151.06	175.36	150.67	185.06	216.65
IS SRC D	159.06	171.27	153.87	170.92	184.62
IS OPT A	182.36	188.96	167.52	195.96	201.76
IS OPT B	211.18	204.15	196.13	238.62	249.69
IS OPT C	170.37	161.89	155.92	182.32	212.76
IS OPT D	177.90	162.92	160.51	190.15	200.45
VC SRC A	182.03	164.81	155.66	169.60	176.26
VC SRC B	184.74	201.02	145.73	200.41	204.09
VC SRC C	170.83	168.06	142.50	181.75	187.39
VC SRC D	153.16	168.03	138.10	181.78	178.55
VC OPT A	189.73	182.05	174.48	195.09	189.02
VC OPT B	160.98	165.73	151.27	165.91	196.21
VC OPT C	161.56	172.23	152.11	186.78	226.65
VC OPT D	164.77	165.83	162.41	193.54	205.73

Table A.18: Mean Degree of selections, all factors

Conditions and Group	**D1**	**D2**	**GR**	**NS**	**Rnd**
IS SRC A	1.731	2.089	2.234	1.969	1.847
IS SRC B	1.668	1.907	2.337	1.992	1.922
IS SRC C	1.547	2.057	2.244	2.108	1.897
IS SRC D	1.686	2.035	2.187	1.928	1.950
IS OPT A	1.793	2.173	2.371	2.187	2.089
IS OPT B	1.494	2.073	2.174	1.968	1.774
IS OPT C	1.657	1.965	2.309	2.142	1.909
IS OPT D	1.728	1.979	2.341	2.128	2.017
VC SRC A	2.631	2.375	2.454	2.204	2.787
VC SRC B	2.991	2.697	2.502	2.422	3.064
VC SRC C	2.574	2.427	2.218	2.296	2.864
VC SRC D	2.268	2.431	2.258	2.243	2.768
VC OPT A	2.764	2.644	2.887	2.522	2.929
VC OPT B	3.080	2.773	3.122	2.812	3.819
VC OPT C	2.628	2.633	2.485	2.419	3.301
VC OPT D	2.748	2.596	2.655	2.448	3.204

Table A.19: Mean number of adjusted Moves, all factors

Conditions and Group	**D1**	**D2**	**GR**	**NS**	**Rnd**
IS SRC A	18.63	20.82	17.41	16.60	24.85
IS SRC B	20.97	21.22	23.52	29.05	21.11
IS SRC C	19.11	18.15	22.66	23.93	21.51
IS SRC D	17.70	17.57	22.37	23.28	24.44
IS OPT A	18.23	18.03	22.57	17.63	17.87
IS OPT B	13.87	14.87	12.86	16.25	12.43
IS OPT C	15.63	16.43	17.43	14.83	15.40
IS OPT D	16.33	19.48	21.43	17.63	15.39
VC SRC A	31.06	39.41	35.61	32.37	48.16
VC SRC B	15.07	28.49	32.16	31	26.41
VC SRC C	29.84	41.03	32.70	19.64	26.98
VC SRC D	35.08	36.61	29.10	28.81	26.57
VC OPT A	15.90	25.10	21.60	22.07	23.2
VC OPT B	11.87	21.73	15.10	14.5	13.43
VC OPT C	16.37	21.73	28.37	23.20	19.67
VC OPT D	16.40	27.77	27.63	23.67	27.56

Table A.20: Performance, search, and cognitive load measures for each Graph Type

Measure	D1	D2	GR	NS	Rnd
PAO	5.998	9.991	13.119	9.688	11.957
Moves	19.46	24.23	23.80	22.11	22.80
Undos	8.221	12.34	11.72	11.77	13.47
Time per Move	2.339	2.476	2.284	2.674	2.447
Toggles	3.296	6.837	6.288	6.159	7.007

Table A.21: Performance, search, and cognitive load measures for each Graph Type, separated by Problem Version

Condition	Measure	D1	D2	GR	NS	Rnd
Search	Undos	12.06	16.26	15.27	15.92	16.38
Optimization	Undos	4.43	8.47	8.21	7.66	6.43
Search	Time per Move	2.13	2.35	2.00	2.48	2.36
Optimization	Time per Move	2.55	2.60	2.47	2.87	2.43
Search	Toggles	4.30	8.73	8.38	8.63	9.48
Optimization	Toggles	2.31	4.95	4.20	3.69	4.55

Table A.22: Performance, search, and cognitive load measures for each Graph Type, separated by Problem

Condition	Measure	D1	D2	GR	NS	Rnd
Independent Set	PAO	7.952	11.967	10.112	10.880	12.762
Vertex Cover	PAO	4.045	8.015	16.277	8.497	11.152
Independent Set	Undos	4.95	8.17	7.60	10.39	8.12
Vertex Cover	Undos	11.54	16.56	15.92	13.18	14.66
Independent Set	Moves	17.57	18.34	20.00	19.92	10.11
Vertex Cover	Moves	21.38	30.17	27.66	24.34	26.54
Independent Set	Time per Move	2.34	2.59	2.35	2.81	2.57
Vertex Cover	Time per Move	2.34	2.36	2.22	2.53	2.32
Independent Set	Toggles	2.62	5.51	5.36	5.24	6.40
Vertex Cover	Toggles	3.98	8.16	7.22	7.11	7.57

A.4.3 Model Figures

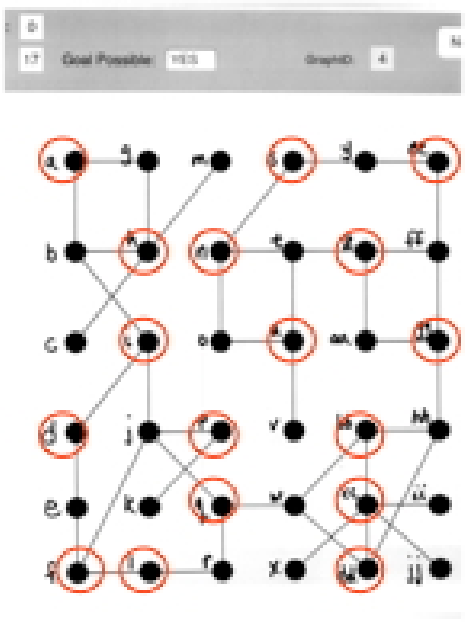


Figure A.51: Subject s17's solution to instance 4

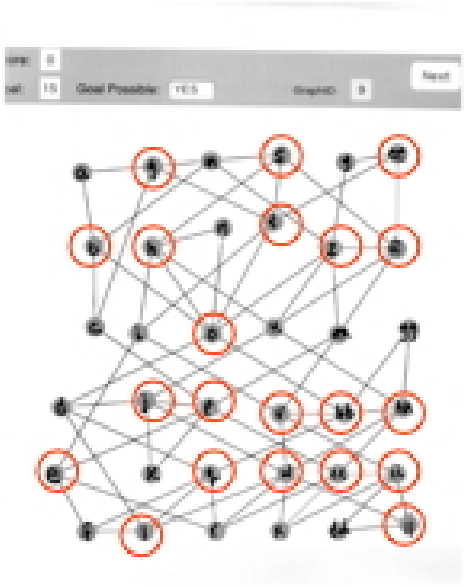


Figure A.52: Subject s17's solution to instance 9

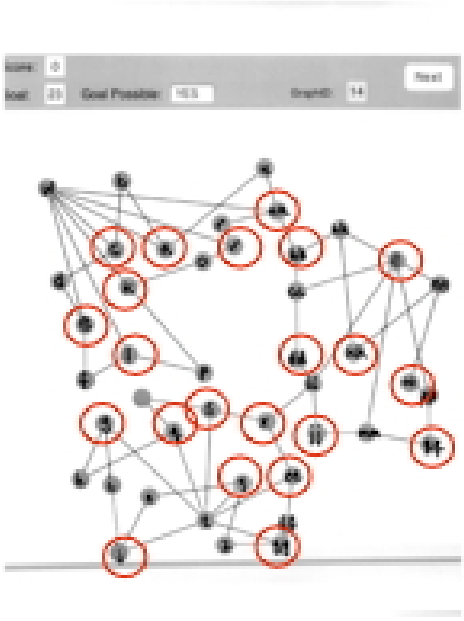


Figure A.53: Subject s17's solution to instance 14

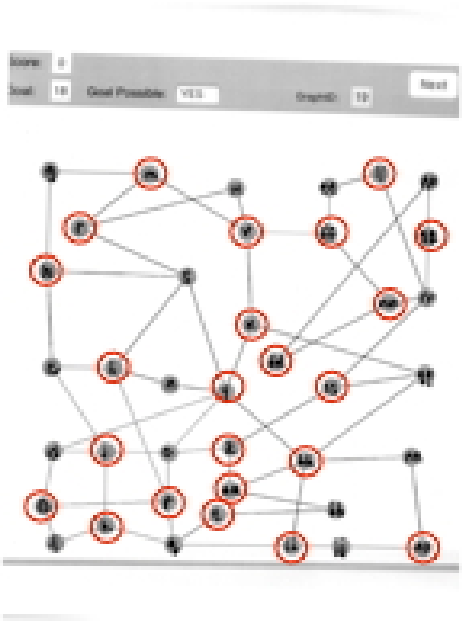


Figure A.54: Subject s17's solution to instance 19

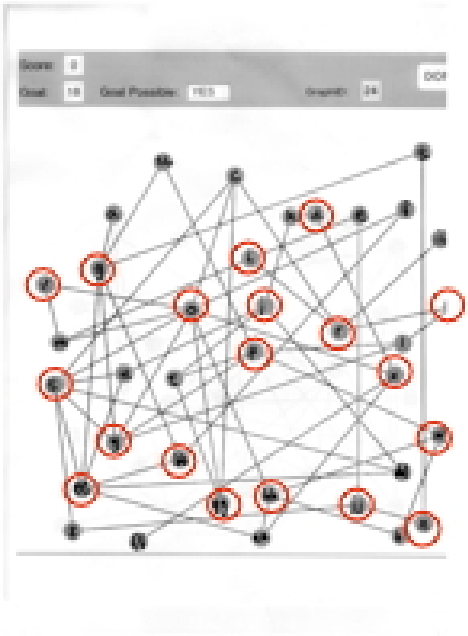


Figure A.55: Subject s17's solution to instance 24

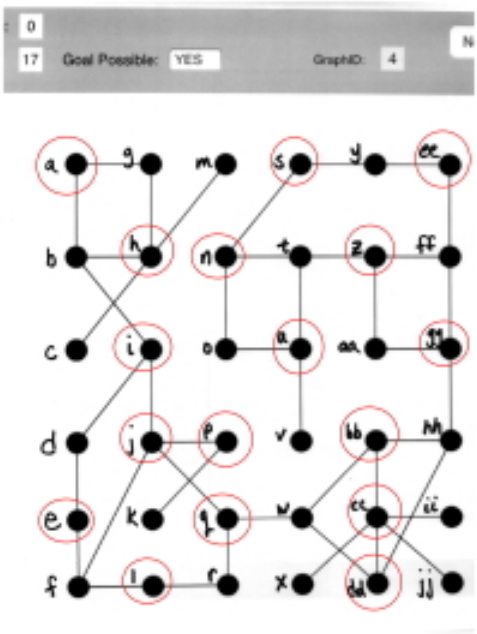


Figure A.56: Subject s20's solution to instance 4

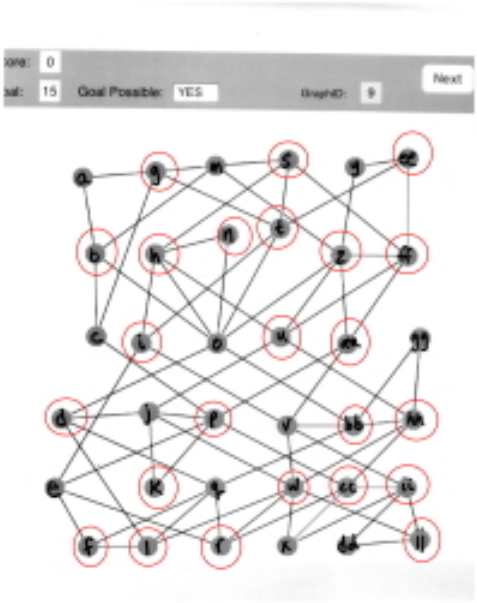


Figure A.57: Subject s20's solution to instance 9

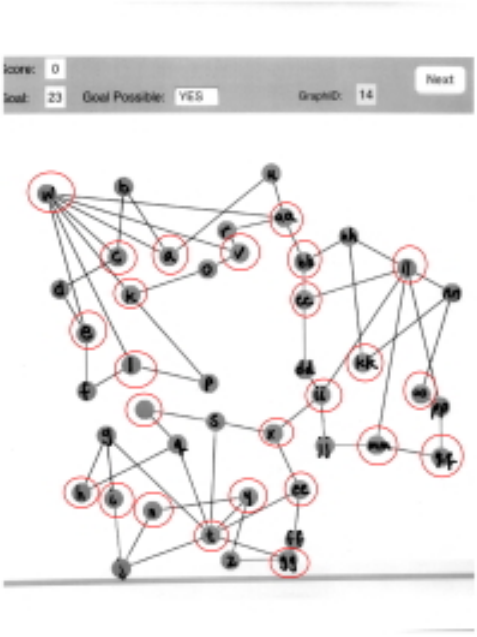


Figure A.58: Subject s20's solution to instance 14

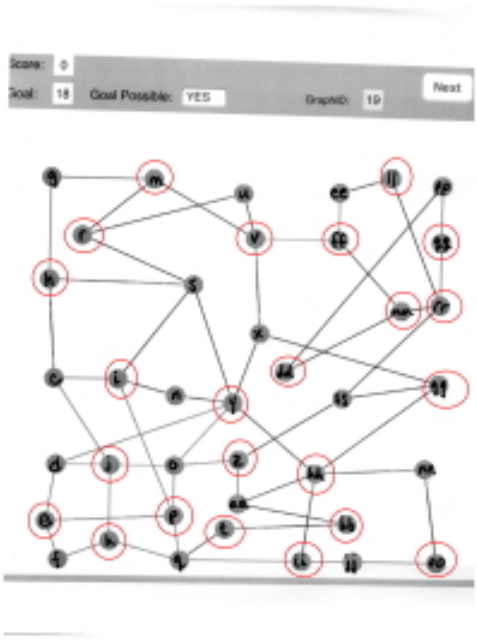


Figure A.59: Subject s20's solution to instance 19

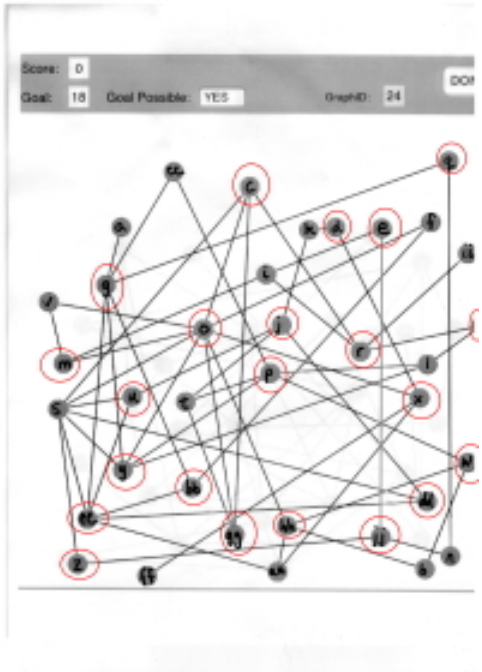


Figure A.60: Subject s20's solution to instance 24

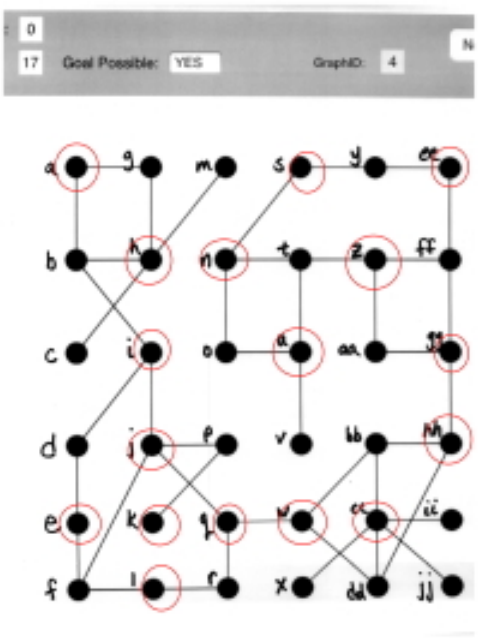


Figure A.61: Subject s84's solution to instance 4

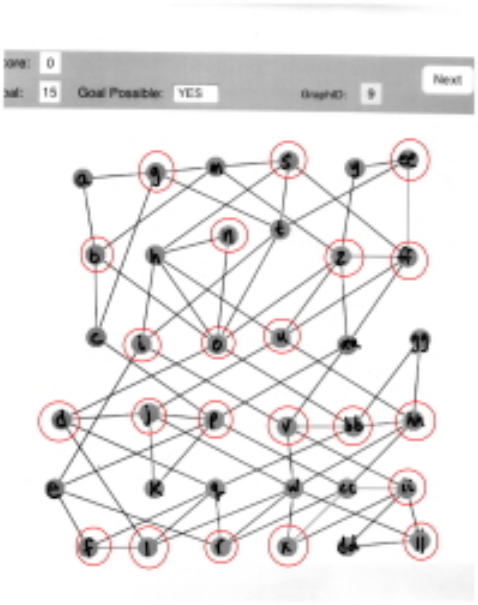


Figure A.62: Subject s84's solution to instance 9

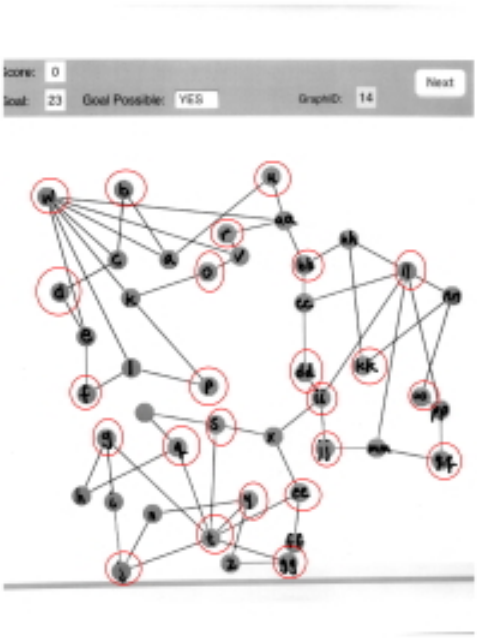


Figure A.63: Subject s84's solution to instance 14

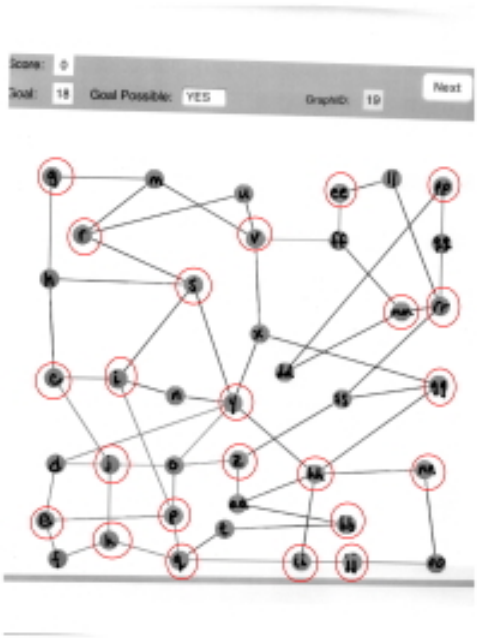


Figure A.64: Subject s84's solution to instance 19

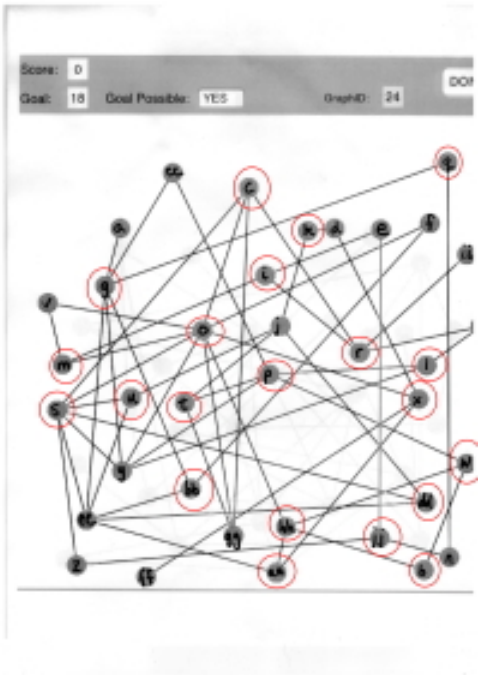


Figure A.65: Subject s84's solution to instance 24

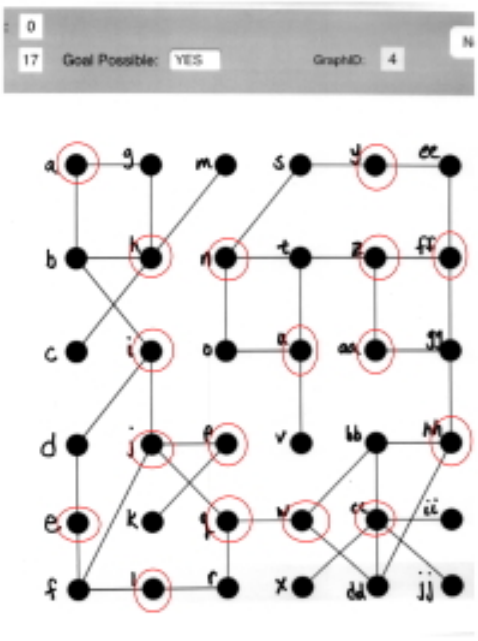


Figure A.66: Subject s5's solution to instance 4

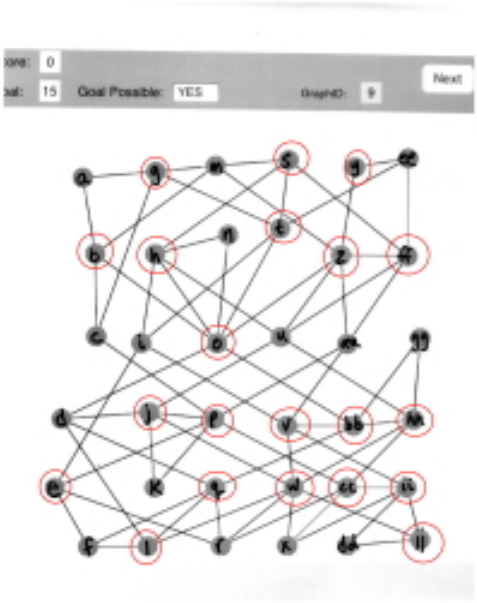


Figure A.67: Subject s5's solution to instance 9

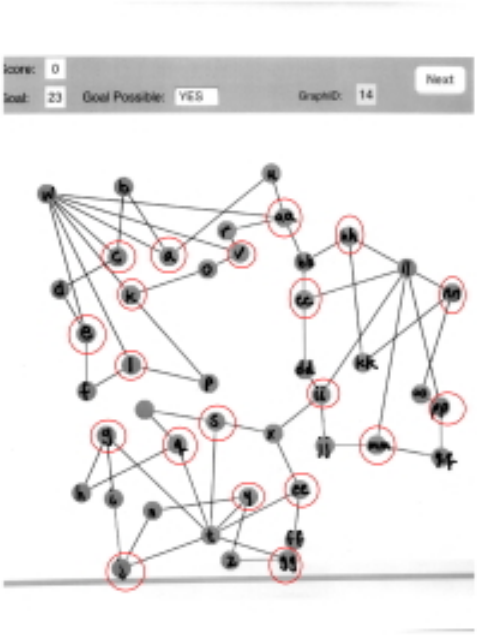


Figure A.68: Subject s5's solution to instance 14

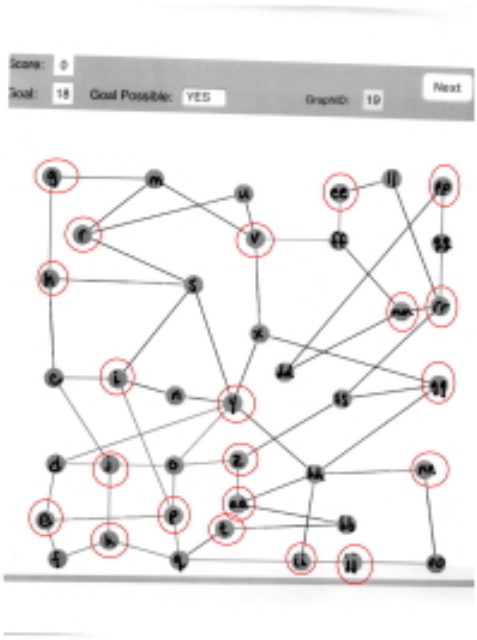


Figure A.69: Subject s5's solution to instance 19

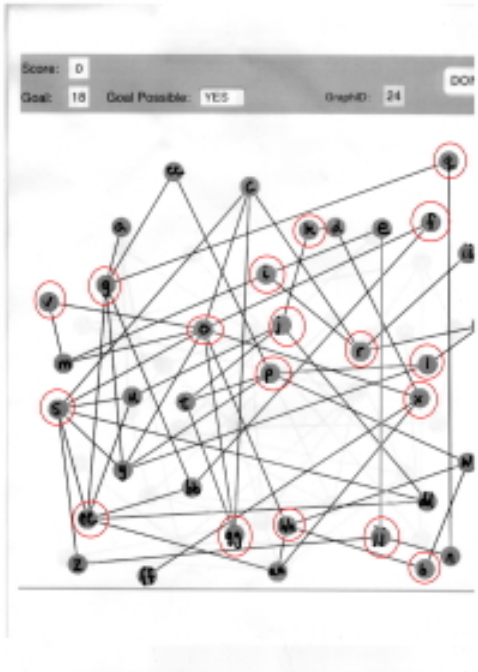


Figure A.70: Subject s5's solution to instance 24

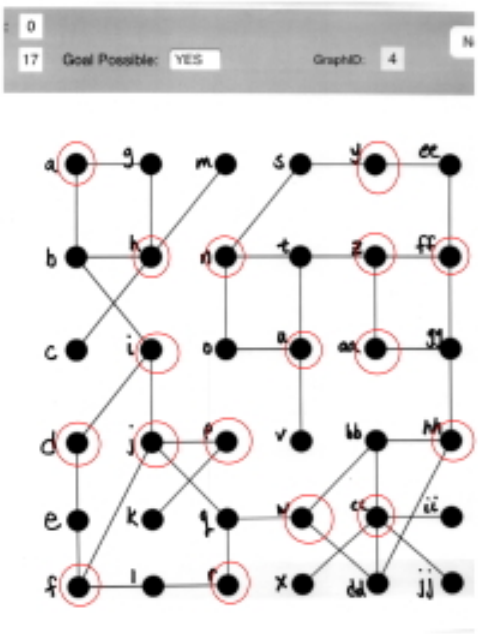


Figure A.71: Subject s22's solution to instance 4

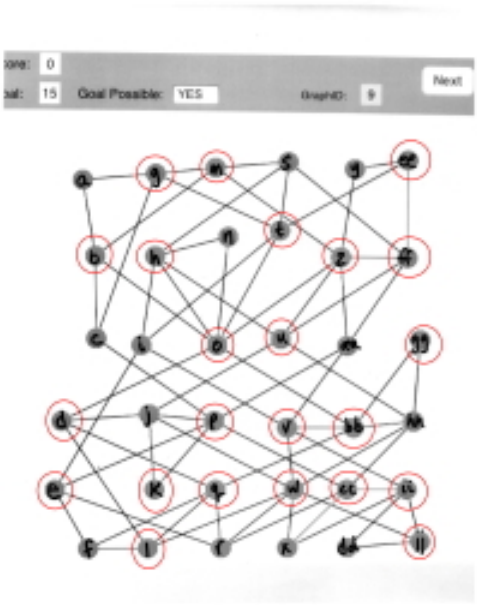


Figure A.72: Subject s22's solution to instance 9

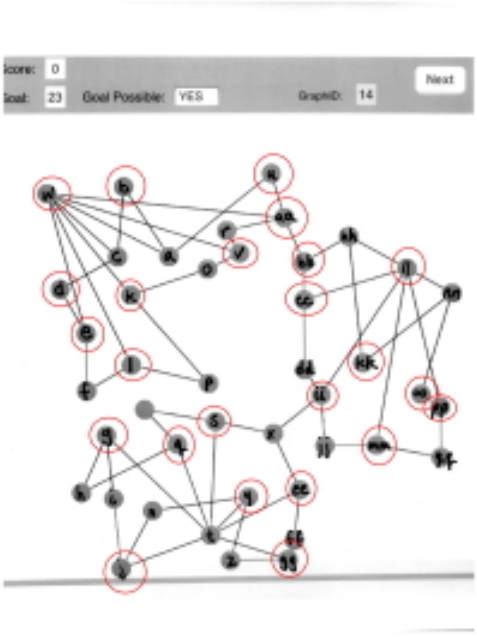


Figure A.73: Subject s22's solution to instance 14

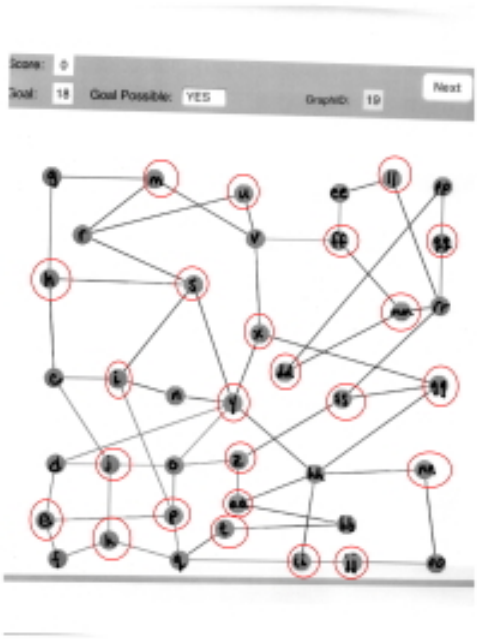


Figure A.74: Subject s22's solution to instance 19

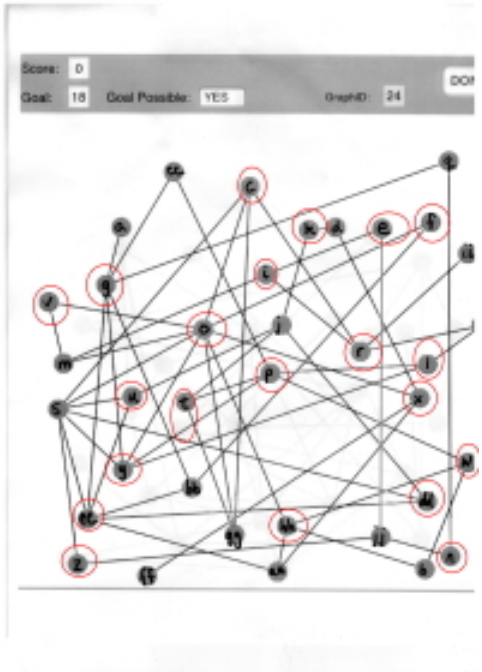


Figure A.75: Subject s22's solution to instance 24