Understanding Problem Solving and Collaboration in Open-Ended Environments

Aaron Bauer

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Zoran Popović, Chair

Steven Tanimoto

Daniel Weld

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Understanding Problem Solving and Collaboration in Open-Ended Environments

Aaron Bauer

Chair of the Supervisory Committee:
Professor Zoran Popović
Computer Science & Engineering

Countless human pursuits depend upon creative problem solving, especially in complex, open-ended domains. As technological support for doing this kind of work in online digital environments grows, an opportunity exists to create a new generation of intelligent problem-solving systems. These environments have the possibility of actively guiding and facilitating individual and collaborative problem solving toward the most productive outcomes. They could scaffold effective solving strategies for novices, intervene in the solving process to suggest areas of focus, or take the form of layers of machine intelligence that schedule individual and group work and dynamically adapt environmental parameters to increase solution quality.

Few of these innovations will be possible, however, without a deep understanding of the problem-solving process in the domain of interest. Such an understanding would need to address the full space of strategies solvers employ, how they fit together and change over time, and how they contribute to both success and failure. In this dissertation, I investigate individual and collaborative problem-solving behavior in open-ended environments to address the question *what makes groups and individuals successful problem solvers?*

First, I present an general framework for automatically extracting patterns of problem-solving behavior from user actions. This framework addresses constructing multivariate time series from logging data on user actions, recursively clustering these time series to produce fine-grained patterns of behavior, and selecting the model with the most understandable set

of patterns. I evaluate this framework on three domains: the scientific-discovery games *Foldit* and *Mozak* and the real-time strategy game *Starcraft II*. These evaluations demonstrate the generality of this technique as well as how the extracted patterns illuminate high-performing behavior.

Second, I develop a visualization-based analysis to identify patterns in *Foldit* users' problem-solving structure. I describe the design of a domain-specific visualization of the problem-solving process in *Foldit* that balances preserving the complexity of the data and producing a tractable representation of behavior. I use these visualizations to identify patterns relating to exploration, optimization, and the use of automated tools. Analysis of how these patterns differ between high- and lower-performing users indicates that successful problem solvers explore more broadly and more frequently avoid local minima.

Finally, I address a suite of questions concerning collaboration in *Foldit*. I investigate how social systems in *Foldit* impact individuals, finding evidence that collaboration has a positive effect on both participation and performance. Next, I explore factors associated with group performance, and find that measures of collective and individual skill have a strongest correlation with group performance. Lastly, I present an ontology of team structures in *Foldit* and show that the amount of collaborative refinement is far more predictive of team performance than parallel exploration or team size.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank the many people who helped make this dissertation possible. My PhD advisor Zoran Popović for eight years of guidance and support. My committee member Steve Tanimoto for his valuable feedback and excellent seminar. My committee members Dan Weld and Andy Ko for their insights. My undergraduate research advisor, Rónadh Cox, for giving me the best possible introduction to research. My formal collaborators Jeff Flatten, Seth Cooper, Eric Butler, Kyle Thayer, Nell O'Rourke, Whitaker Brand, and Stuart Reges. My informal collaborators Josh Gardner, Rahul Banerjee, Greg Nelson, Yun-En Liu, Adam Smith, and Erik Andersen. My fellow Center for Game Science grad students Zuoming Shi, Yvonne Chen, and Alex Jaffe. My CGS colleagues Roy Szeto, Colin Bayer, Ed Paradis, Yanko Yankov, Ric Gray, Dmitri Danilov, Jenny Vogel, and Ourania Abell. Marianne Lee and Barbara Krug for their help on Dragon Architect. The developers and researchers working on *Foldit* and *Mozak*, especially Brian Koepnick. My dear friends Ellen Stuart, Jake Levinson, Kristin Siu, Steve Rutherford, Matt Mullen, James Jaffe, Karl Lapo, Kaleena Fraga, Nick Arnosti, and Antal Spector-Zabusky. My parents Peter and Janet and my brother Gordon.

I would also like to acknowledge the organizations that have funded my work including the National Institutes of Health grant 1UH2CA203780, RosettaCommons, Amazon, National Science Foundation Grant No. 1629879, Grant No. DRL-1639576, and Grant No.: DGE-1546510, the Oak Foundation Grant No.: 16-644, DARPA grant FA8750-11- 2-0102, and the Bill and Melinda Gates Foundation.

Chapter 1
INTRODUCTION

Open-ended, creative problem solving is integral to virtually every domain of human endeavor from education, to policy making, to scientific research. As the speed and sophistication of digital technology has increased, more and more of these problem-solving tasks are digitally mediated, perhaps incorporating digital tools or taking place entirely within a digital environment. In education, this often consists of more targeted, more adaptable, or more scalable ways of delivering traditional content, such as intelligent tutoring systems or massive open online courses (MOOCs). Domains like policy and research, meanwhile, have seen crowd-based systems such as *The Climate CoLab* [31] and *Foldit* [12] that offer entirely new approaches to long-standing problems.

The next generation of digital problem-solving systems present a tremendous opportunity to introduce a wide array of supports, interventions, and augmentations into the solving process. This opportunity is especially promising in creative, open-ended domains where good solutions are not known. For students or novices, systems could scaffold known effective solving strategies or prompt reflection on what worked and what did not after a task is complete. Interventions could take the form of suggested moves or areas of focus, diagnostics of users' current solution or solving process, or carefully chosen example solutions (which research on design teams suggests can have a significant impact on solution quality [23]). There is also a large space of possible augmentations, particularly in the form of additional information and automated tools. A user could receive visualizations of what other users have tried or how their own exploration of the problem space has progressed as Robison explored with the CoSolve Consultant [63]. Depending on the task, important actions (such

as local optimization) or feedback (such as real-time evaluation of solution quality) could be provided via automation. Finally, layers of machine intelligence could be created to schedule individual and group work and dynamically adapt environmental parameters such as team size to achieve increased speed and solution quality, as has been applied to task routing in Wikipedia [13]. Figure 1.2 provides an extended example of the various ways an intelligent problem-solving system could intervene in the context of learning to program.

Many of these innovations are only possible given a deep understanding of the problem-solving process. Such an understanding must (1) explain the strategies users employ and show how different techniques fit together in a solving process, (2) address the evolution and adaptation of these strategies over time and in response to tasks with different properties, and (3) account not only for instances of success, but also for inefficient or ineffective solving processes, such that they may be identified and corrected. In addition, as collaboration is an increasingly integral part of problem solving, from scientific discovery [93] to management [26], a deep understanding must extend to both individual and group settings. Only with analytical techniques able to account for all of these pieces can we realize the full potential afforded by digital problem-solving environments.

Furthermore, is will be vital for such techniques to include aspects specific to the problem domain of interest. Substantial research on problem solving has found that experts use more efficient, domain-specific strategies (called *strong* strategies), while novices usually rely on *weak*, domain-independent strategies [55, 9, 36]. Especially for open-ended, or *ill-structured*, problems, high-impact supports or interventions will almost certainly need to be rooted in domain-specific knowledge, either to scaffold the acquisition of that knowledge or guide users toward the most effective strategies. A general theory of problem solving, while a very valuable goal in its own right and the subject of much attention in the literature, is unlikely to support the kind of innovations digital problem-solving environments offer to make possible.

Problem solving is a very broad term and applies to a huge variety of activities. In this dissertation, I am focused on problem solving in *open-ended environments*. By this I mean contexts without known good solutions, where complex demands are made of problem

solvers and sophisticated tools provided to them, and that lack clearly defined procedures for navigating the problem space. These properties align with long-standing ideas of *ill-structured* problems in the literature. In their typology of problem solving, Jonassen describes *design problems* as "among the most complex and ill-structured kinds of problems that are encountered in practice" [37]. The properties of design problems include a *vague goal with few constraints*, *answers that are neither right or wrong, only better or worse*, and *limited feedback*. The ill-structured nature of these open-ended problems necessarily deprives us of the structures, such as clear goal states and straightforward relationships between intermediate states and goal states, that typically form the basis of existing detailed and quantitative analyses of problem-solving behavior. We need new analytical techniques that can apply in these uniquely challenging settings if intelligent problem-solving systems are to play a productive role in helping users tackle ill-structured problems. Hence, the central question animating this dissertation: *what makes group and individuals successful problem solvers in open-ended environments?*

1.1 Foldit *as an Open-Ended Problem-Solving Environment*

The scientific-discovery game *Foldit* presents an ideal environment for studying individual and collaborative behavior on real-world open-ended problems. By modeling the functions of proteins, the workhorses of living cells, *Foldit* challenges its users to resolve the shape of proteins as a 3D puzzle. These puzzles are completely open and often under-specified, sharing many of the properties Jonassen attributes to design problems. *Foldit* provides only a vague goal (i.e., find a good configuration of the protein) and feedback is limited to a noisy estimation of solution quality (i.e., real-time feedback is limited to a single numerical score corresponding to the protein's current energy state, and users must frequently progress through many low-scoring configurations to reach a good solution). Developing users from novices to experts capable of overcoming these difficulties and solving protein structures currently unsolved by scientists is central to *Foldit*'s scientific-discovery community. Solutions generated by *Foldit* users have led to three results published in the journal *Nature* [12, 19, 40]. The ill-structured

nature of the problems it poses and its objective of state-of-the-art biochemistry results make *Foldit* a highly suitable setting in which to study problem solving on open-ended problems, and data from *Foldit* is used throughout this dissertation.

At present, *Foldit* is a one-of-a-kind system. Its long history (active since 2008), large, dedicated user base, and highly complex protein-folding task enable analysis of collaboration and problem-solving behavior on open-end problems at a scale beyond any other existing dataset. Thus, there is great potential value in developing analytical techniques to understand behavior in *Foldit*. Studying *Foldit* could yield unique insights as well as enable future problem-solving systems to follow its lead. Furthermore, problem solving in *Foldit* has broadly applicable properties. *Foldit* users make successive modifications to a solution, and can return to previous solutions to try a different approach. Collaboration in *Foldit* focuses on handing off solutions from one user to another, each making their own improvements. At the individual and collaborative level, *Foldit* shares its basic structure with many design and analysis tasks, and results gleaned from its wealth of data have potential implications well beyond crowd-sourced protein folding.

To provide useful background for the following chapters, I provide a general overview of *Foldit* here. It presents users with a 3D representation of a protein and tasks them with manipulating it into the lowest energy configuration. Each protein posed to users is called a puzzle. Users' solutions to each puzzle are scored according to their energy configuration, and users compete to produce the highest scoring results. Figure 1.1 shows the *Foldit* interface.

Users have many tools at their disposal when solving *Foldit* puzzles. They can manipulate and constrain the structure in various ways, employ low-level automated optimization (e.g., a *wiggle* tool makes small, rapid, local adjustments to try and improve the score), and trigger solver-created automated scripts called *recipes* that can programmatically use the other tools. Previous work analyzing user behavior in *Foldit* has focused primarily on recipe use and dissemination [11] and recipe authoring [39], though there has been limited discussion of group solving dynamics [12].

Foldit has several different types of puzzles for users to solve. In this dissertation, I

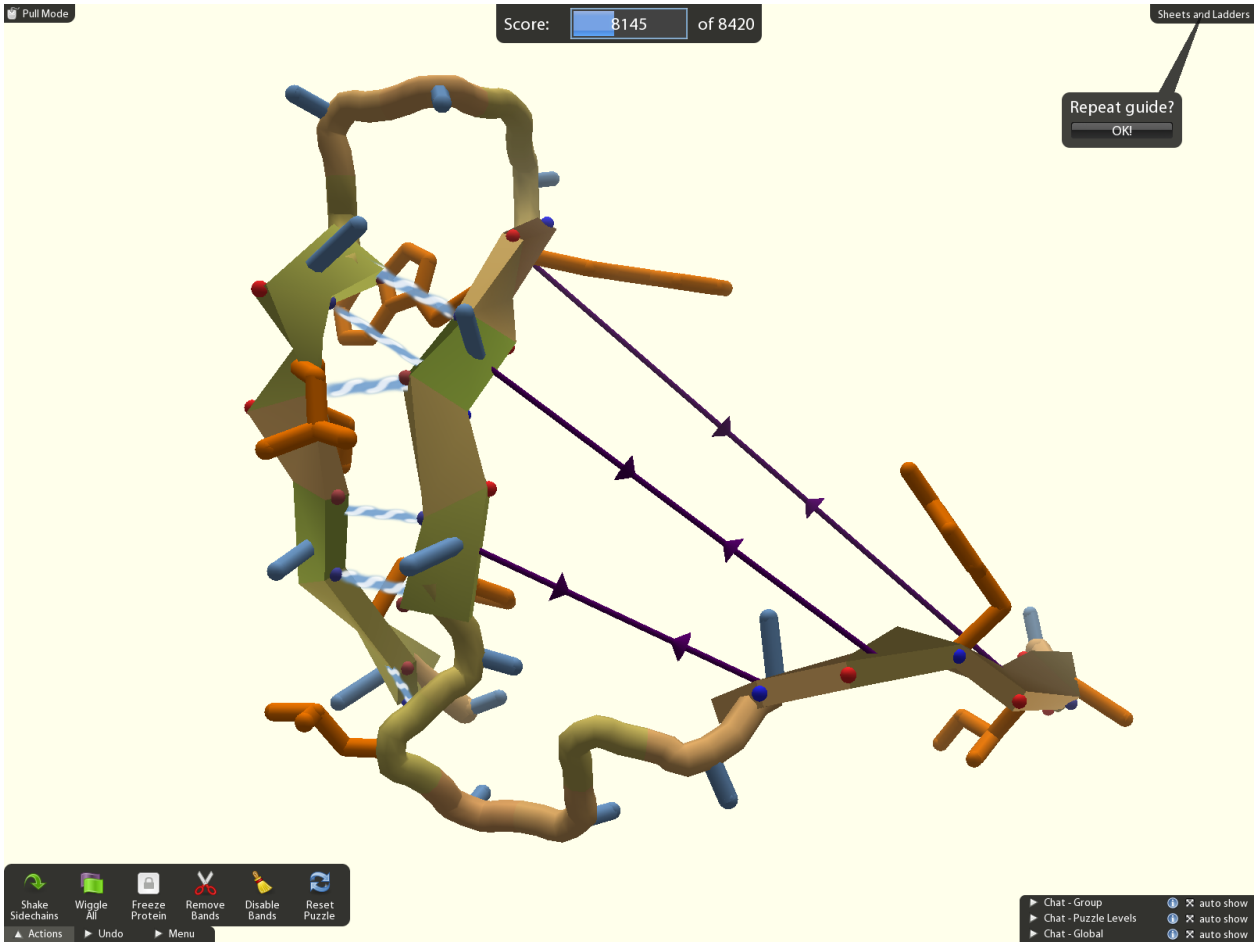


Figure 1.1: The *Foldit* interface. *Foldit* solvers use a variety of tools to interactively reshape proteins. In this figure, a user uses rubber bands to pull together two sheets, long flat regions of the protein.

focus on the most common type of puzzle, *prediction* puzzles. These are puzzles in which biochemists know the amino acids that compose the protein in question, but do not know how the particular protein folds up in 3D space. This is in contrast to *design* puzzles in which users insert and delete which amino acids compose the protein to satisfy a variety of scientific goals, including designing new materials and targeting problematic molecules in diseases.

In addition to generating solutions individually, *Foldit* users can join together in groups to tackle puzzles collectively. While users can communicate via a variety of typical online tools

such as text chat and image sharing, *Foldit* provides an explicit mechanism for collaboration called *evolving*. Whenever a member of a group generates an individual solution (called a *soloist* solution), they can choose to share it with the other members of their group. Those other members can then import this soloist solution directly into their client, and attempt to modify and improve it. If a teammate successfully improves on the soloist effort, the new, improved solution is recorded as an *evolver* solution. A group's official solution for a puzzle is the highest scoring solution, soloist or evolver, produced by any member of that group.

1.1.1 Definition of Success

An analysis of problem-solving behavior in *Foldit* necessarily requires a definition of success. The lack of a metric for solution quality comparable across puzzles, however, makes it difficult to quantify individual and group success over time. The only absolute metric of solution quality is the score computed from the protein's current configuration, but since this metric is contingent on various structural properties of the protein, there is no reliable baseline to use to compare solutions dealing different proteins. Interpretation of the best scores, as well as the differences in score among a set of solutions cannot easily generalize across multiple puzzles. Hence, a *relative* rather than absolute metric is the natural choice for tracking success in *Foldit* over time.

An obvious relative metric, a simple ranking of solutions by score, fails to capture any notion of puzzle difficulty or the degree to which top-scoring solutions exceed the competition. That is, if a puzzle is relatively easy, and there are dozens of similarly-scoring solutions, a ranking-based metric will treat this situation the same way it would treat a puzzle where one or two solutions dramatically outscore everything else. Motivated by these considerations, I use *the ratio of a solution's score to the score of the best solution* as my measure of performance in *Foldit*. In other words, the performance of an individual or group on a given puzzle is measured as the ratio of their score to the best soloist or group solution, respectively. This metric accounts for the failings of ranking: if all the top solutions have similar scores, they will all have very similar performance; if one solution significantly exceeds the rest, my metric

for performance will reflect this.

1.2 Dissertation Outline

The rest of this dissertation proceeds as follows. In Chapter 2, I discuss prior work across a variety of subfields related to the work presented here including problem solving systems, problem-solving behavior, and collaboration. In Chapter 3, I present *pattern extraction*, a general technique for automatically extracting patterns of behavior from user actions. The core idea behind pattern extraction is to treat user behavior as a multivariate time series and perform time series clustering to identify temporally coherent patterns in that behavior. I describe the pattern extraction architecture that lays out the three components of this technique: (1) constructing a single multivariate time series from logs of user behavior, (2) recursively clustering this time series to extract fine-grained patterns, and (3) a model selection algorithm for selecting the clustering that produces the most understandable set of patterns. I then evaluate pattern extraction on three problem-solving domains to demonstrate its generality and ability to identify high-performing behavior. For *Foldit* and another scientific-discovery game *Mozak* (for background on *Mozak*, see Section 3.2), I visualize the surprising diversity of behavior among the top echelon of users, identify the patterns associated with increased performance beyond the benefits of generic additional effort, and use patterns to predict user performance. I perform a proof-of-concept evaluation on the real-time strategy game *Starcraft II* and show how patterns can illustrate a fundamental tradeoff users face.

In Chapter 4, I take advantage of the inherent tree structure of problem solving in *Foldit* to develop a visualization-based analysis. I devise a domain-specific visualization that balances preserving the complexity of the data and producing a tractable representation of behavior. I use this visualization to identify six patterns of user behavior involving exploration, optimization, and the use of automated tools. A comparison of how these patterns are used by high- and lower-performing users indicates that successful problem-solvers explore more broadly and more effectively avoid local optima. Chapter 5 turns from diagnosing individual problem solving to examining collaboration in *Foldit*. I show that early experiences with

Foldit's two social systems, competition and collaboration, are both associated with increased participation. I then design a synthetic controlled experiment to assess the effect of joining a group on individual performance, finding a moderately positive effect. Next, my exploration of how features at multiple scales relate to group performance finds that measures of individual and collaborative skill have strong correlation with performance, that participation has moderate correlation, and that individual and group experience have weak correlation with performance. I follow this with an ontology of team structures in *Foldit* and observe that more complex structures tend to achieve better performance. Lastly, I design eight features of team structure and use them to predict team performance. Examining the role of each feature within the model shows that team structures facilitating more collaborative refinement do more to increase performance than those emphasizing more parallel efforts.

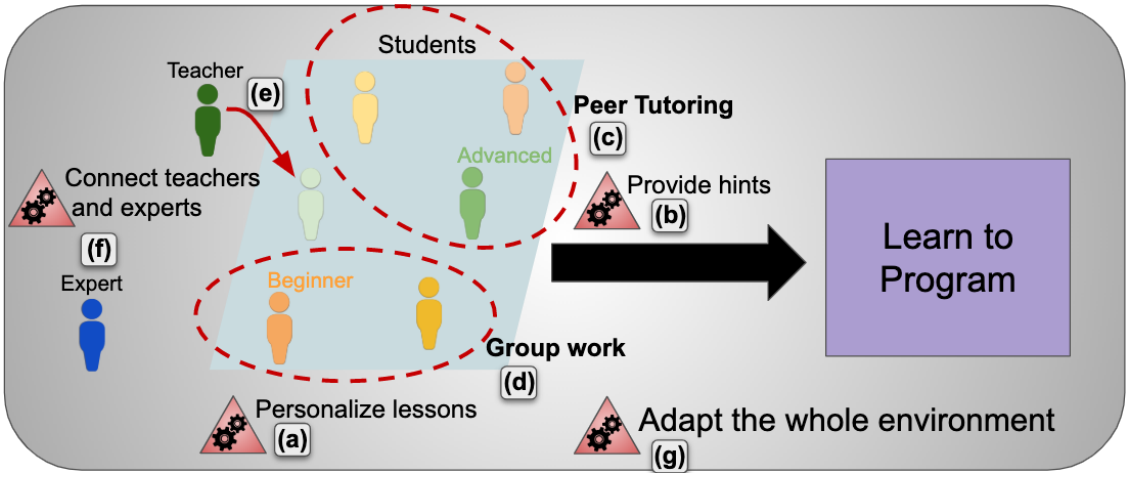


Figure 1.2: This figure illustrates how an intelligent problem-solving system could intervene to help users learn to program. Though this dissertation focuses on less structured problems, learning to program is a complex task and provides a useful concrete example of the kind of problem-solving system motivating my work. There are different types of users in this hypothetical system: students at varying levels of capability (six figures in pale blue section) from beginners to the most advanced (shaded from orange to green), teachers, and domain experts (anyone with relevant expertise such as curriculum specialists or professional software developers). **(a)** Students can differ in myriad ways: not everyone is at the same point in the curriculum, people learn best in different ways and at different paces. Our system could personalize instructional content and modality to fit these different needs. **(b)** Our system could provide just-in-time hints or other guidance to help students avoid getting stuck and to correct key misconceptions. While existing systems such as intelligent tutoring systems are capable of both (a) and (b) to some extent, features like (c)–(g) would significantly advance the state-of-the-art. **(c)** Our advanced student has just mastered a module on class inheritance, but two of their peers are still working through it. Instead of pushing the advanced student further ahead of the class or letting them sit idle, our system sets them up as a peer tutor for their classmates still studying class inheritance, accelerating the learning of the group as a whole. **(d)** The system recognizes that the beginner student has mastered conditionals, but still struggles with for loops, while another student has mastered for loops, but makes mistakes with conditionals. As a result, it pairs these two students on a group work exercise that involves both conditionals and for loops, enabling them to bring each other up to speed. **(e)** Another student has demonstrated a grasp of the basics of recursion, but struggles to understand situations with multiple recursive calls. Our system goes to the teacher and highlights this specific roadblock, suggesting this would be a good time for one-on-one instruction on this specific item. **(f)** Our system could also coordinate domain experts and teachers. It is not uncommon for K-12 computer science teachers to have deep experience with teaching, but less subject matter expertise. Hence, they might benefit from being connected with a professional software developer or other expert participating in the system. **(g)** Finally, our system could adapt environment-wide parameters as a class progresses. Perhaps initially, the system does not make a distinction between integer and floating point data types, and only introduces this complexity once certain milestones are met. All of these features would depend on understanding what makes people successful, what skills separate an expert and a beginner, and how the system can help users realize their full potential. It is also important to note that if we shifted to a more open-end task, such as designing better social media platforms, there would be many more unknowns and ambiguities in how the system might usefully intervene. Thus, it is necessary to develop new techniques for understanding problem solving in open-ended domains to enable problem-solving systems to support work on complex, real-world tasks.

Chapter 2
RELATED WORK

In this chapter I discuss three broad areas of prior work relevant to this dissertation: digital problem solving systems, approaches to studying problem-solving behavior, and approaches to studying collaborative problem solving.

2.1 Problem-Solving Systems

Citizen science environments like *Foldit* and *Mozak* have long been a fertile subject for the study of online communities [32]. As these efforts have moved online, their contributions have spanned many fields including biology [78], environmental studies [41], and astronomy [50]. In their typology of citizen science, Wiggins and Crowston identify *virtual* projects as an important emerging type with high capacity for motivating continued participation in scientific research [89]. Even among virtual or *investigation* types of citizen science (i.e., those focused on generating scientific knowledge as opposed to driving civic action or educating users), the vast majority do not involve complex problem-solving tasks like those in *Foldit*. Instead, users are engaged in data collection, such as contributing observations of birds in eBird [78], or classification, such as labeling images of galaxies by type in Galaxy Zoo [50]. Hence, problem solving is little studied in the citizen science domain, and *Foldit* presents an opportunity to expand the field in that direction.

Regardless of their complexity, many citizen science tasks are highly collaborative. This, along with the scale at which virtual citizen science efforts can operate, has made them useful settings for researchers interested in understanding the dynamics of these collaborative spaces. For example, Rotman et al. studied the motivations that led both scientists and volunteers to engage in collaborative projects together [66]. They highlight the importance of feedback

mechanisms in sustaining participation, such as the timing of motivational probes and the availability of information about how the data generated by the volunteers will be used. Though *Foldit* is the source of data for much of my analysis, this dissertation does not focus on the dynamics of citizen science per se. Instead, the properties of the *Foldit* environment make highly suitable for studying a complex collaborative problem-solving ecosystem, and I am motivated by questions applicable to many real-world collaborative problem-solving settings.

Finally, collaborative problem solving has been studied in the context of evaluating systems designed to facilitate novel mechanisms of collaboration. For example, CoSolve [22] allows users to both pose and solve problems, visually representing the solving process as state-space search trees, The Climate CoLab [31] combines model-based planning, online debates, and electronic voting to enable collaborative development of plans to address climate change, and CrowdForge [45] presents a general framework for crowdsourcing complex tasks such as article writing. This work demonstrates the broad and exciting space of possibilities for digital problem-solving environments, and suggests there remains a lot of unexplored design space. The work presented in the following chapters supports further innovation in this area.

2.2 Problem-Solving Behavior

The study of problem solving stretches back well over a century. Robertson identifies three philosophical perspectives that defined the study of problem solving behavior for much of its history [62]. The *behaviorist* approach focuses on a cause-effect model of problem solving, often conceptualized in terms of stimulus and response. Researches such as Thorndike [83], Watson [87], and Skinner [74] operationalized problem solving as sequences of learned *habit mechanisms* chosen or found through trial and error in response to stimuli. *Gestalt* psychologists like Koffka [47] and Wertheimer [88] take a differing view, arguing that problem solving is best understood not through the atomized, procedural approach of the behaviorists, but rather as the result of a holistic understanding of a problem's structure. Wertheimer termed this *productive thinking*, in contrast to the *reproductive thinking* involved

in blindly following a procedure.

The appearance of the programmable computer, along with advances in other fields, spurred the third of these perspectives, *information processing*. First proposed by Newell, Shaw, and Simon [56], this approach aims to explain human problem solving in terms of simple information processes. That is, the mostly unconscious mental encoding, storage, retrieval, and manipulation of information. This perspective (e.g., [57, 25, 72]) underlies much of the subsequent work on problem solving. My formulation in Chapter 4 of problem solving in *Foldit* as a search through a problem space follows from these theories. In contrast, I take a more data-driven approach in Chapter 3 and frame problem-solving as an unstructured series of actions. Unlike the foundational work referenced here, my overall contribution is not a general theory of all human problem solving, but rather new tools for understanding specific problem solving behaviors in ill-structured domains such as *Foldit* and the insights these methods produce.

One major thread in research on problem solving with particular relevance to my objective of understanding what contributes to success in open-ended environments is the study of the differences between novice and expert problem solving. Pitt's fine-grained analysis of strategic behavior when solving standard chemistry problems is an example that those studying behavior in digital environments should aspire to [59]. They specified a set of 24 problem-solving *subroutines* such as *listing assumptions*, *extracting patterns from data*, and *defining the goal state* and then used these subroutines to code transcripts of solvers' verbal protocols. Pitt lays out how subroutine use differed with solver experience, concluding that novices defined problems poorly and showed limited ability to coordinate information or operations. Creativity will be required to replicate this sort of analysis in digital domains without the benefit of verbal protocols, but this type of approach could greatly aid the development of a deep understanding of the problem-solving process.

Expert use of domain-specific knowledge and strategies compared to weaker, more general techniques used by novices is a clear theme to emerge from work in this area. Larkin et al. found that on physics problems novices rely on general strategies such as *means-ends* (i.e.,

choosing actions based only on the differences between the current state and the goal state), while experts use more efficient strategies that rely on their domain knowledge [48]. Similarly, in mathematics Sweller, Mawer, and Ward show how solvers transition from means-ends to *forward-chaining* (a bottom-up approach using domain knowledge to select relevant principles and then applying those principles to the givens of the problem) as they gain experience [79].

This trend persists across a very broad range of educational disciplines as shown by Groen, Patel, Schultz, Lochhead, and others in [75]. Though the use of domain-specific strategies is part of all the observed novice-expert differences, it is telling that both the specific differences, and the methodology necessary to elucidate those differences, varies across domains. For example, in medicine these differences only become apparent on explanation tasks rather than more direct problem-solving tasks because the response modes are highly verbal, while in physics there is a clear distinction between novices and experts in their ability and willingness to draw and use schematic diagrams. This domain variance bolsters my argument that if we are to enable intelligent problem-solving systems, we need techniques for understanding problem solving that account for domain-specific nuance. Finally, Jefferies et al. extend this line of research to the non-educational context of software design [33]. Their analysis finds that novices often perform only a shallow decomposition of the larger problem, and do little generation and evaluation of alternative solutions, whereas experts make extensive use of decomposition and exploration of alternatives in their solving process.

This prior work reinforces the importance of understanding the differences between novices and experts in any study of the problem-solving process. This is especially true in *Foldit*, where the environment not only formulates proteomics problems such that players can solve them, but also must cultivate the necessary expertise in a community where many members have no prior biochemistry knowledge. My findings on the differences in behavior between high- and lower-performing solvers in *Foldit* and *Mozak* are consistent with the consensus in the literature that experts' knowledge allows them to effectively use strategies that are poorly or infrequently used by less-skilled solvers. I also contribute a granular understanding of the specific behaviors and differences at work in these scientific-discovery domains.

As a significant factor in the novelty of my work is extending the analysis of problem solving in digital environments to ill-structured domains like *Foldit*, it is important to acknowledge previous work on constructing models of solving ill-structured problems. Sinnott [73] and later Jonassen [36] propose general models of this process that include problem formulation, generating possible solutions, and the evaluation, implementation, monitoring, and adaptation of solutions. In contrast, I contribute methods for a granular, domain-specific understanding of problem-solving behavior. It remains an important aspect of future work to bridge this gap and investigate whether granular behaviors are consistent with the 30,000-foot perspective of general models.

2.2.1 Visualization

Visualization is a prominent modern approach to understanding behavior, problem-solving and otherwise, in digital environments. Games are one type of environment well-suited to visualization. Wallner and Kridlstein provide a thorough review of visualization-based analysis of gameplay data, highlighting diagrams, heatmaps, movement visualizations, self-organizing maps (a technique for clustering players by features of gameplay), and node-link representations as the primary categories [86]. The latter category is by far the most appropriate for analyzing progress through a solution space without an explicit spatial correspondence (i.e., without a game world or map). In fact, building on the *Playtracer* system developed by Andersen et al. [2], Liu et al. generate visualizations of gameplay on an introductory puzzle in *Foldit* [49]. While the authors draw some insights about the design of the puzzle from the visualization, they acknowledge the very high dimensionality of *Foldit*'s solution space makes it infeasible to successfully apply *Playtracer* to *Foldit* puzzles in general.

Eagle et al. produce more structured visualizations of problem-solving behavior in educational games and intelligent tutoring systems called *interaction networks* [18]. These networks offer insight for hint generation and a flexible method for visualizing student work in rule-using problem solving environments. Similar to my approach, the authors consider both visual interpretation of the networks as well as computational features of the graph

representation. Like *Playtracer*, interaction networks are not applicable to complex, open-ended settings. High dimensionality challenges modeling each user action as an edge in the network and complex action spaces undermine the reliance on a state-matching function to produce a concise visualization. Nevertheless, these systems demonstrate promise and broad applicability of a visualization-based approach, and I build upon this work by extending visualization-based analysis to the more complex domain of *Foldit*. Unlike them, I do not represent the behavior of many players simultaneously in a single visualization or develop a general visualization technique, but instead develop a rich, domain-specific visualization of individual solving behavior. My approach depends on careful tailoring of the visualization to the *Foldit* domain, leveraging the full space of available visual encodings to enable identification of patterns of problem-solving behavior.

2.2.2 Statistical Analysis

Researchers have also attacked the mountains of data produced by educational games and tutoring systems using a variety of statistical and machine learning techniques. Straddling both visualization and statistical methods, Horn et al. use a mixed-methods approach to analyze the behavior of nine individuals playing a minimum-spanning-tree-based puzzle game [29]. The authors use hierarchical clustering to elucidate different playstyles and visualize player progressions to understand where they encounter points of difficulty. Also using clustering, Tóth et al. investigate strategy use in *MicroDYN*, a tutoring environment with tasks based on linear structural equations [84]. In particular, the authors examine the feature vectors of the cluster centroids and identified four groups of students according to what degree each student used an effective vary-one-thing-at-a-time strategy. Through a combination of automated detectors, path analysis, and classroom studies, Rowe et al. investigated the relationship between a set of six strategic moves in a Newtonian physics simulation game and performance on pre- and post-assessments [67]. They found that the use of some moves mediated the relationship between prior achievement and post-test scores. Harpstead et al. instrumented *RumbleBlocks*, a game for teaching basic principles of stability,

to enable full replay from gameplay logs [28]. They apply this *Replay Analysis Engine* to compute features of student solutions and use them to assess in-game learning and the alignment of game levels with learning objectives.

Falakmasir et al. describe the *SPRING* system that ingests log data from an educational game and produces a simple, interpretable model of in-game actions that can predict learning outcomes [21]. Provided the data conforms to the *slot and filler* format common among gameplay logs, SPRING discretizes player actions via clustering and then trains two Hierarchical Dirichlet Process Hidden Markov Models to model the actions of high-performing and low-performing students (that students can be accurately separated into these two categories based on assessment data is an assumption SPRING makes). SPRING is shown to predict geometry post-test scores based on student data from the educational geometry game *Alice in AreaLand* with moderate accuracy. The authors highlight that SPRING does not rely on domain knowledge engineering as its primary advantage, and suggest that the HMMs it produces form an interpretable model of player behavior. Though they use decision trees instead of HMMs, Malkiewich et al. also propose to glean insight from the models produced by a machine learning algorithm [53]. Using features derived from data from the educational physics game *Physics Playground*, the authors study which features were included (and the signs of the coefficients on those feature) in the decision tree trained to classify *gold badge* solutions (i.e., highly efficient solutions) and the tree trained to classify *silver badge* solutions (i.e., correct, but less efficient, solutions). They find sub-optimal performance characterized by exploratory, tinkering behavior, while top performers tend to have previous exposure to a problem and take more efficient actions.

What separates my contributions from this prior work is my focus on complex, open-ended domains. Environments like *Foldit*, unlike the systems studied in the prior work I have described here, does not have simple known best strategies (e.g., vary-one-thing-at-a-time) nor established correct answers (e.g., gold badges). Like many of the above approaches, the pattern extraction method I present in Chapter 3 uses a form of clustering to understand user behavior. A key difference, however, is that the large action spaces and completely

open-ended tasks of the targeted problem-solving domains present a far more challenging clustering problem that the relatively constrained user behavior in an educational game. The novel analytical framework I describe facilitates the application of time series clustering to extract useful patterns of user behavior in these challenging domains.

2.3 Collaboration

As befits a task of such ubiquitous importance, collaborative problem solving has been long studied in a variety of ways. Malone and Crowston present a useful interdisciplinary framing for work in this area [54]. Writing over two decades ago when digital environments were in their infancy, they asked *how will the widespread use of information technology change the ways people work together?* They argue that a vital part of what they call the emerging field of *coordination theory* is understanding how people collaborate in new environments and how new environments can be structured to facilitate effective collaboration. *Foldit* is one of these new environments to have been enabled by increased capability for distributed work on difficult and complex problems. Hence, my work in Chapter 5 on collaboration in *Foldit* advances this field as conceived by Malone and Crowston by bringing collaborative problem-solving behavior in this new environment under close and rigorous scrutiny.

Such scrutiny is needed, as existing models of collaboration have uncertain relevance in a domain like *Foldit*. Furthermore, the traditional in-person, small-scale approaches to the study of collaboration are not applicable to size and scope of activity in *Foldit*. For example, highly-cited work such as Barron's close analysis of coordination among two groups of three sixth-grade boys solving math problems [3] and Roschelle and Teasley's microanalysis of a single pair of solvers working together on an activity in a physics simulation [64] elucidate fine-grained collaborative behavior in settings and with methods ill-suited to transfer to an environment such as *Foldit*.

One modern evolution of Malone and Crowston's coordination theory (and, in fact, the route Malone and his lab has taken) is the concept of *collective intelligence*. First described by Woolley et al., collective intelligence is defined analogously to individual intelligence as the

general ability of the group to perform a wide variety of tasks [91]. Collective intelligence is a property of a group itself, distinct from the abilities of individual group members. Woolley et al. find in two studies that collective intelligence (measured via factor analysis of group performance on a suite of standard tasks) is far more predictive of group performance on a number of different tasks than intelligence scores for individual group members. Kim et al. extend collective intelligence to the domain of competitive online games [42]. Using a set of 11 tasks called the Test of Collective Intelligence [20], the authors shows the collective intelligence of five-person teams in the online game *League of Legends* predict teams' competitive performance.

While collective intelligence is a promising approach to the study of collaboration, as it currently stands, it has little to say about the processes of group interaction that actually contribute to high performance. Woolley, Aggarwal, and Malone identify this limitation and point to the understanding of these processes as a ripe area for future work [90]. My work helps meet this deficit in the context of *Foldit* by taking a detailed look at the features that contribute to group success at multiple levels. The results from the study of collective intelligence suggest that this is an important complement to the analysis of individual behavior and abilities.

Another perspective on collaborative problem solving is the consideration of team structure, including which individuals make up the team and how they relate to each other throughout the problem-solving process. Team structure gains particular salience for the emergent ad-hoc teams common to digital environments like Wikipedia or *Foldit* that depend on the ad-hoc collaboration of strangers. These environments may imposing more formal structure on collaboration (e.g., the soloist-evolver model in *Foldit*) than found in traditional teams. Thus, the design choices for how these formal structures function are potentially of critical importance to the long-term success of problem-solving systems. The terminology in the literature on team structure, often published in the Management field, does not align perfectly with the digital crowd work context. Structural factors relevant in an organizational context such as *interdependence* or *team self-leadership* (e.g., [77]), are not necessarily well-suited to

ad-hoc digital teams. What management literature calls *team design features* are a much better fit for the structural considerations of collaboration in domains like *Foldit*. My analysis of team structure and team performance in *Foldit* only somewhat matches the features Stewart's meta-analysis identifies as having a positive relationship with performance [76]. While features of *Foldit* team structure important for predicting team performance such as the depth of collaboration could reflect the value of intrateam coordination, team size, and heterogeneity Stewart finds, my work is only a first step in bridging our understanding of these two very different contexts. My findings are also somewhat consistent with recent studies of team structure in competitive online games. Like Pobiedina et al., I identify prior experience collaborating with current teammates as a positive factor in performance, but unlike in their analysis of the game *Dota 2*, I do not find it to be the most important factor [60]. Both Pobiedina et al. and Cheng et al. found teams employing a diversity of roles perform better in games where each team member selects a specific character to play [8]. While my analysis does not contradict this, it remains to be shown whether findings about diversity in environments with explicit role selection carry over to other contexts.

Robison performed a similar analysis of fine-grained collaborative structures in the context of the CoSolve system [63]. Like the features of team structure I use to predict team performance in Section 5.6, Robison designed metrics to measure turn taking, the use of various annotations, and the distribution of work across the solution tree structure. The power of this analysis of CoSolve is limited in the conclusions it can reach, however, by the available data of six teams of three. In contrast, *Foldit* offers a one-of-a-kind dataset (2,454 teams formed from 880 users), allowing more robust analysis. Nevertheless, Robison's finding that giving some teams access to detailed diagnostic information about their own collaboration had no clear observable effect is an important data point when considering how future problem-solving systems might intervene in collaboration.

2.3.1 Crowd Work

Of all the digital contexts in which collaboration is studied, crowd work has received perhaps the most attention. This research spans tasks of all kinds, but the study of crowds working on complex, open-ended tasks are of primary interest here. Collaboration in crowd work settings is often similar to collaboration in *Foldit*, as both are ad-hoc and user driven. Several studies have examined the factors contributing to success in these environments. Cranshaw and Kittur use a combination of data analysis and visualization similar to my approach to study the principles behind the success of the *Polymath Project*, a group of mathematicians who collaborate online to solve open mathematics problems [15]. They find that broad participation is very valuable, even if a small percentage of the users created most of the content, and that the leadership of a couple of users was vital in guiding the project to its mathematical success. The authors also identify several barriers to entry that might discourage newcomers from participating, and suggest that better structuring and linking of content could make it easier for new users to identify important content and understand what tasks need attention.

Another online mathematics project broadly similar to *Foldit*, *MathOverflow*, focuses on collaboratively solving novel mathematical *micro-problems*. Tuasczik, Kittur, and Kraut investigate collaboration on these real-world open-ended problems and develop a simple taxonomy of collaborative acts [81]. They build regression models of the impact these various acts have on solution quality. While these models show that the collaborative acts, such as clarifying the question, critiquing an answer, and revising an answer, increase solution quality, they do not yield any further insight into how successful collaboration functions in *MathOverflow*. In a radically different domain, Settles and Dow conduct an exploratory study of an online songwriting community, investigating the factors contributing to successful ad-hoc collaboration [70]. The authors combine four years of server logs from February Album Writing Month (FAWM), an online music event, with a web survey of over 200 members of the FAWM Facebook group. Using regression models and qualitative survey data, they find

communication, shared interests, and status are key predictors of whether two individuals will collaborate, and once a collaboration is formed, that equal division of labor appears to be the most significant factor in perceived success.

My work complements this existing body of research by extending the study of online collaboration to the scientific-discovery game context. The differences in my approach from those I describe here are driven by differences in both the *Foldit* domain and the data available in *Foldit* logs. Most significantly, I do not include any text content produced by *Foldit* solvers in my analysis, instead relying solely on actions taken when interacting with the solutions themselves. While text communication may form an important component of collaboration in *Foldit*, I am able to discover useful insights from examining how solvers contribute to shared solutions.

Chapter 3

PATTERN EXTRACTION FROM ACTION SERIES

3.1 Introduction

A major strength for digital problem-solving systems is the opportunity actively intervene in the problem-solving process in real time. Historically, this has taken many different forms, from providing feedback in an intelligent tutoring system to informing collaborators on their team's progress with a tool like the CoSolve Consultant [63]. The possibilities are vast: just-in-time interventions to help users overcome roadblocks or catch mistakes; rapidly scaffolding new users by demonstrating expert techniques or pairing them with other learners; and enhancing team performance by bringing together users with the right mix of specialties. Part of the incredible potential of future problem-solving systems lies in increased capabilities to improve and coordinate users.

Many of these interventions are only possible, however, if the problem-solving system has detailed and reliable measures of users' skill. That is, measures that can identify the specific behaviors relevant to success in a way that can generalize across many users. For example, to scaffold expert techniques for new users, the system must both identify coherent examples of the desired expert behavior and identify the absence of such behavior among the targeted new users. The scaffolding could be synthesized by the system itself, or created by a domain expert. In the latter case, the system would need to be able to present the domain expert with a comprehensible representation the behavior in question.

As the scaffolding example suggests, one compelling way to frame measures of user skill is to ask the question *what do high and low performers do differently?* This framing dovetails with the overarching question animating this dissertation: *what makes individuals and groups successful problem solvers?* In this chapter, I will present a novel automated and general

technique for extracting patterns of problem-solving behavior and show how these patterns can be used to identify behavior that sets high performers apart from other users. While the immediate motivating application for identifying these patterns is to explain differences between high and low performers, their long term purpose is to support the kind of problem-solving system interventions described in Chapter 1. These kind of interventions, as well as controlled experiments to establish causal relationships between problem-solving behavior and outcomes, are resource-intensive in terms of time, implementation effort, and human subjects. In complex open-ended domains, however, the curse of dimensionality makes it impractical to use interventions as the initial analytical approach. Hence, a technique like the one I present here is a necessary first step for identifying dimensions for potentially impactful intervention.

The core premise of this *pattern extraction* technique is to learn labels for user behavior over time. These labels apply to subsets of a user's problem-solving process and correspond to specific patterns of behavior that may be shared by many different users. For example, a given user might start out doing pattern A, before transitioning to a different behavior, pattern B. As input, pattern extraction takes sequences of user actions. These actions could correspond one-to-one with user interface interactions, represent combinations of interactions, or represent some higher-level action inferred from lower-level interactions. This form of input makes pattern extraction potentially highly general, as user behavior in many contexts can be characterized as a chronological sequence of actions.

To extract patterns from these input sequences (i.e., to label subsets of these sequences as belonging to various patterns), I first represent them as multivariate time series where different types of actions are distinct streams in the time series. I then perform a clustering on this time series that generates a cluster label for each time step, and view the learned clusters as different patterns of behavior. Figure 3.1 shows a schematic representation of the output of the pattern extraction process.

Discovering these patterns of behavior has many potential applications within a problem-solving system. To deliver a just-in-time intervention, the system could use the information

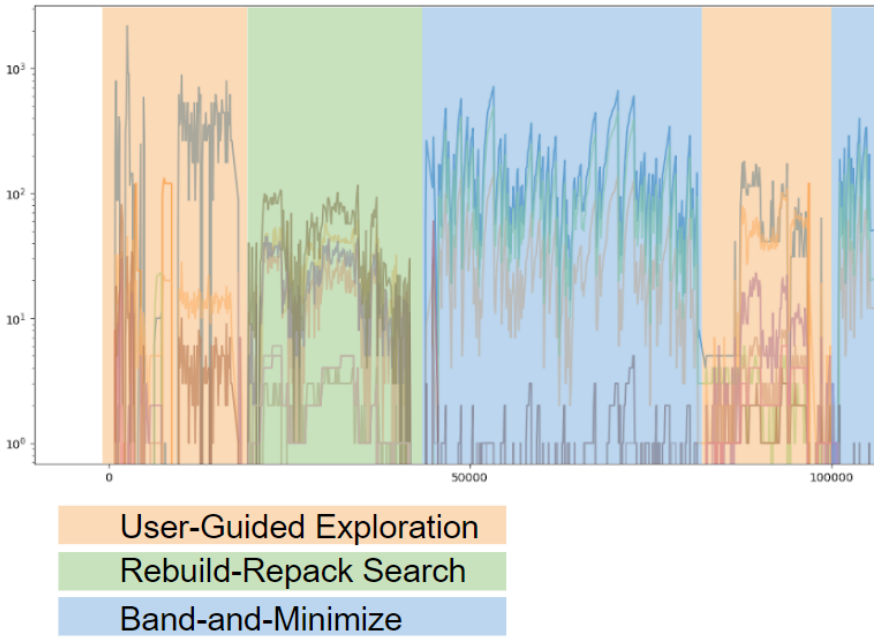


Figure 3.1: This figure shows a schematic representation of the output of the pattern extraction process. The underlying plot is a portion of an action series for a *Foldit* user. Each of the colored lines shows the use of a particular type of action, with time on the x-axis and amount the action was used on the y-axis. The colors overlaid on the plot demonstrate the kind of segmentation pattern extraction aims to produce. Pattern extraction determines that the user began in the orange pattern, switched over to the green pattern, and then alternated between the blue and orange patterns. Annotations giving these patterns domain-specific meaning (e.g., the orange pattern is *user-guided exploration*) are provided post-hoc by a domain expert.

of what pattern a user is in and empirically what pattern they should transition to next to get the best results. To train novices to become more like experts, the system could identify both what patterns a novice already employs as well as specific expert patterns to teach them. Finally, to coordinate team effort, a system could use patterns to assess specialties (i.e., what patterns each team member employs most effectively), in order to know which aspect or phase of the problem team members are best suited for.

The primarily data-driven method I describe here stands in contrast to traditional qualitative methods for understanding expertise. I see these approaches as highly complementary,

as each can provide unique insights into the problem-solving process. In particular, when designing problem-solving system interventions it will be important to incorporate both quantitative evidence of effective behavior, like that provided by pattern extraction, and a qualitative understanding of how users view the system and potential active interventions.

Pattern extraction does hold, however, some key advantages over qualitative methods. First, as a general and automated technique, pattern extraction has more flexibility to apply to new or little-studied domains. The logistical effort involved of extending pattern extraction to a new data source will thus typically be substantially less than what a qualitative method requires in terms of identifying relevant users, interviewing them, and coding the transcripts. Second, qualitative methods face far more constraints in terms of adaptation and sample size than pattern extraction. A data-driven method can identify when a situation changes or evolves, and is easier to update in response, whereas qualitative methods remain a snapshot in time without the laborious collection of new data. Similarly, qualitative methods can easily overfit to a small sample. Pattern extraction reveals substantial diversity among top *Foldit* players (see Section 3.4.1), and a qualitative deep dive would risk missing this diversity and reaching misleading conclusions. Finally, qualitative investigations of expertise necessarily rely on the perceptions of experts. These experts may have biases or be unable to explain their problem-solving process. Hence, a data-driven analysis may be able to go beyond what experts know or can clearly articulate.

The contributions contained in chapter are as follows: (1) an automated and general technique for extracting patterns from user actions, including methodology for constructing appropriate action series, performing the extraction recursively to distinguish fine-grained behavior, and a model selection algorithm designed to maximize interpretability and (2) an evaluation of this technique on three different problem-solving domains. For evaluations on *Foldit* and another scientific-discovery game *Mozak*, I identify high-performing patterns and use the extracted patterns as features in models predicting user performance. I also conduct a proof-of-concept evaluation of pattern extraction on the real-time strategy game *Starcraft II*, and show that patterns can capture a fundamental economic-military tradeoff.

3.2 Background

Related Work At its core, pattern extraction is a clustering of multivariate time series data. This genre of analysis is widespread across many disciplines from biology to finance. As Aghabozorgi et al.'s broad review of this work shows, however, relatively little of this analysis has been directed towards understanding human behavior, let alone problem-solving [1]. Pattern extraction brings time-series clustering to bear on problem-solving data, and indicates there could be substantial benefit to educational and user-focused domains in looking to the rich literature in time series analysis.

Considering the mining of behavior patterns from temporal data more generally, there is a growing body of work using these techniques to answer questions in education and problem solving. Multiple studies have used behavior data streams (i.e., clickstream data) to investigate educational settings like Massive Online Open Courses (MOOCs). For example, Chen et al. used MOOC clickstream data to learn dependencies between content segments [7] and Cicchinelli et al. used interactions with an online platform to study students' self-regulation strategies [10]. Tang et al. use a process of categorizing and filtering log data from a MOOC course not unlike that described in Section 3.3.1 to design input features for LSTM models of what course resource students will interact with next [80]. In domains like *Foldit* and *Mozak*, users act via a range of sophisticated actions with distinct functions, making it valuable to analyze behavior in terms of these actions rather than raw clicks.

Prior work has also used sequence mining to identify high- and low-performing patterns of behavior. For teams of students navigating relevant and irrelevant information on an interactive tabletop [51] and for students constructing a causal map in a digital tutor [43], researchers discovered short sequences of single actions that distinguished between high- and low-performing participants. In cognitive tutors where the most salient aspects of user behavior are the occurrence and speed of correct and incorrect answer attempts, researchers have used Markov-Model-based analysis to detect students' tactics [71] and problem-solving styles [46]. While these different approaches are useful in these highly-constrained educational

domains, they are far less applicable to complex, open-ended domains.

There are examples in the literature of analysis similar to my work where time series of user interactions are used to identify more complicated patterns of behavior. Boroujeni and Dillenbourg perform data-driven discovery of study patterns among MOOC participants, focusing on a narrow set of interactions relating to when videos and assignments were started [5]. Though this small set of possible interactions makes the specifics of their analysis very different than the pattern extraction technique proposed here, in both cases the interpretation of the discovered patterns and how they relate to outcomes are key components. As another example, Sawyer et al. use student interactions with an educational biology game to create an action trajectory for each student representing the pace of student activity during play [69]. They compare these trajectories with an expert gold standard, finding the more closely a student tracks the expert trajectory, the higher the student's learning gains. I am similarly interested in studying users in relation to expert performance, though my domains of interest lack gold-standard traces of expert behavior. A key difference of my work is a focus on the specific actions and behaviors that are associated with high performance, rather than just the overall trajectory.

Time Series Clustering To perform the time series clustering portion of pattern extraction, I employ the algorithm recently developed by Hallac et al., Toeplitz Inverse Covariance-Based Clustering (TICC) [27]. Given a single multivariate time series and a number of clusters k as input, TICC learns a cluster label for each timestep, as well as a graphical model (in this case a Markov Random Field) representing the structure of each cluster. An additional useful property of TICC is that it attempts to enforce *temporal consistency*, in that adjacent timesteps are encouraged to belong to the same cluster. This is intuitively relevant when clustering problem-solving behavior since it is highly unlikely a human user would rapidly switch between many different behavior regimes in succession.

It is important to note that the use of TICC is not essential to the pattern extraction method. The field of time series analysis is rapidly evolving, and analytical techniques in

problem solving and other domains should evolve accordingly. TICC is well-suited for pattern extraction due to its efficiency (TICC scales linearly with the length of the time series) and temporal consistency objective. The overall architecture of pattern extraction presumes some technique for time series clustering, but does not target TICC specifically beyond the necessary logistical steps to incorporate it into the framework.

Mozak In addition to *Foldit*, this chapter contains an evaluation of pattern extraction on data from the scientific-discovery game *Mozak* [65]. The goal of *Mozak* is to produce full three-dimensional reconstructions of neurons from image data, which are the subject of intense study in neuroscience. To do this, users trace three-dimensional images of neurons. Automated methods typically struggle to correctly trace faint or noisy signal in these images, meaning lots of expensive expert effort is required to produce high-quality reconstructions. The key to *Mozak* is an automated *consensus algorithm* that computes a trace agreed upon by multiple users, allowing non-experts to produce a final result much closer to an expert reconstruction.

Gameplay in *Mozak* is divided up into separate *challenges*, with each tracing a different neuron image. Users have two basic tracing tools: *connect the dots* and *virtual finger*. The former places one trace node at a time and tasks the user with selecting what z-level within the three-dimensional volume of image data the node will be placed at. The latter allows the user to place many nodes at once by clicking and dragging, snapping the nodes to the z-level with the highest signal. Users can also delete parts of their traces as well as undo previous actions. Finally, the user can toggle various overlays including different ways to view consensus, comments on traces in progress left by experts, and automatically generated *hot spots* that highlight areas where additional tracing could improve consensus.

Starcraft II To demonstrate the generality of pattern extraction beyond scientific-discovery games, this chapter also contains a proof-of-concept evaluation on data from the real-time strategy game *Starcraft II* [4]. In *Starcraft*, users compete to conquer each other by harvesting

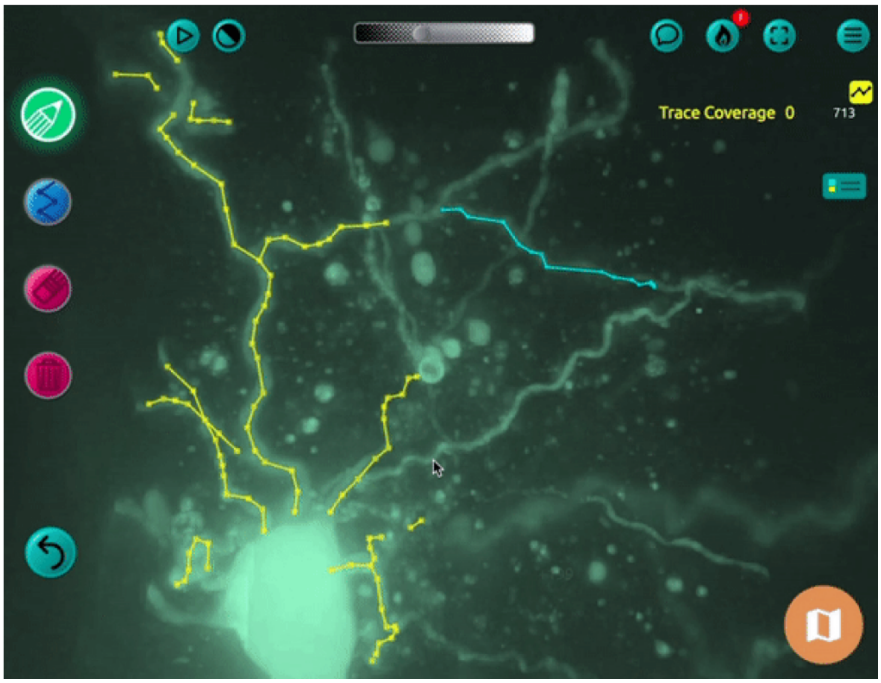


Figure 3.2: This figure shows the *Mozak* interface. The white lines and shapes are the neuron image being traced. The cyan line is the user's trace, while the yellow lines are the current consensus. Available tools are shown along the left side and buttons for altering the view and accessing the menu run along the top.

resources and using them to build armies. Users begin each game by choosing to play as one of three races (Zerg, Terran or Protoss), which determines the military units available to the user. The races are highly asymmetrical and each leads to a different playstyle. *Foldit* and *Mozak* are about focused attention on producing a single good solution. By contrast, a major part of *Starcraft* is managing simultaneous and often conflicting demands on both attention and resources. Users must navigate a tradeoff between investing in economic expansion and building up the necessary military. Militarily, users must judge when to attack and when to assume a defensive posture while expanding their army. Finally, all these decisions are made in real time with incomplete information of the adversary, meaning devoting attention and resources to scouting the opponent is another tension the user has to manage. In my evaluation, I focus on detecting patterns that show changing economic and military emphasis.

While much of the academic work on *Starcraft* has been focused on creating more capable AI agents, there has been some work viewing it as a problem-solving domain. Thompson et al. extracted cognitive-motor, attentional, and perceptual processing measures from *Starcraft* data and found that which measures were most important for predicting skill level changed as skill increased [82]. Yan et al. focused specifically on patterns in the use of army organization and keyboard shortcuts, using them to accurately predict whether a user was a novice or expert [94]. Finally, Ravari et al. developed models to predict the game winner in *Starcraft* based on a combination of time-dependent and time-independent features [61]. The insights produced by this nascent body of work on understanding skill in *Starcraft* demonstrates its potential as a domain for studying human problem solving.

Figure 3.3: This figure shows the *Starcraft II* interface. The majority of the screen is taken up by a top-down view of the battlefield. The user is moving in units from the left to respond to an attack on one of their bases. From left to right along the bottom are the *minimap*, a schematic overview of the whole battlefield, a section showing the currently selected units, and buttons for orders the user can give to the currently selected units. The quantities in the top right show the user's current resource reserves.

3.3 Pattern Extraction Architecture

This section lays out the pattern extraction architecture, including the construction of action series, the recursive clustering to distinguish fine-grained patterns, and an algorithm for selecting the most interpretable model.

3.3.1 Constructing Action Series

The first step in the pattern-extraction process is assembling the input action series. The initial data takes the form of logging data from the target domain (i.e., a list of user interactions). Each item in the logging data is an action or set of set of actions taken by a single user accompanied by a timestamp, a user ID, and session ID, if necessary (e.g., a puzzle ID in *Foldit*). The key decision for transforming this raw data into a multivariate time series is selecting how different user interactions will correspond with variables in the time series.

It is not necessarily the case that every type of action should be its own variable. Some actions may be very closely related and thus can be reasonably combined into a single variable. For example, in *Foldit* adding a band between atoms is logged as a different action than adding a band between residues, but a researcher could place both under a single *banding* variable in the time series. Other actions may not merit inclusion at all. In my iterative development of the pattern extraction technique, I found variables that were active very rarely and only for a few timesteps at a time were not amenable to clustering, and thus added little value while complicated interpretation of discovered patterns. For example, *Foldit* users can manually drag part of the protein into a different position, but this action is typically used only a handful of times on any given puzzle, if at all, and thus is excluded from the time series constructed from *Foldit* data.

The design of the time series variables is a manual process, and a critical point where pattern extraction relies on domain expertise. Some choices for how to map logged behavior to variables will be guided by the semantics of the problem-solving domain, while others will depend on expert judgement. These choices define the space of patterns that can be

extracted, and thus need to be informed by knowledge of what would constitute interesting or meaningful behavior in the target domain. Iteration on these choices will be important for refining the application of pattern extraction to any specific domain.

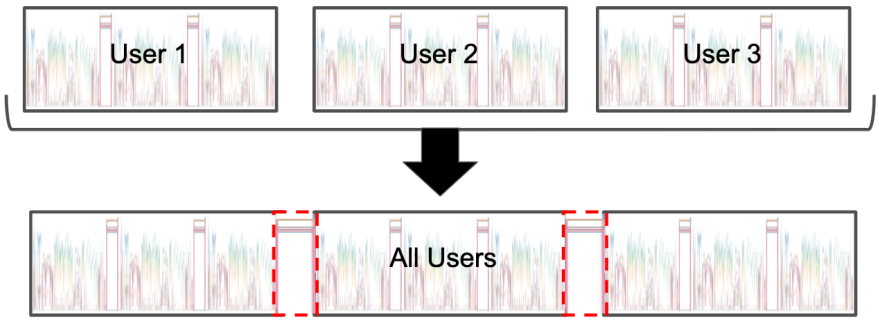
Once the variables for the actions series are designed, the raw data from each user session (i.e., a *Foldit* puzzle, *Mozak* challenge, or *Starcraft* game) can be transformed into an multivariate time series of those variables. At this point, however, there is a mismatch between the input data, many different actions series, and the requirements of the clustering step. The TICC algorithm, as is generally true for time series clustering techniques [96], takes as input a single time series. Pattern extraction could perform clustering separately on the data from each user session, but this would make it difficult to compare the discovered patterns. If the presence and absence of patterns for different users is not easily determined, this undermines the central goal of using patterns to differentiate between high- and low-performers. Thus, pattern extraction must produce a single clustering across multiple sessions and users.

The solution is to combine many different actions series into a single series by connecting them with an extended period of arbitrary noise [1]. If the same distinctive noise is used to connect everything together, the noise will get placed in its own cluster while preventing any of the original data from running together in a misleading way. Figure 3.4 shows first, how data from multiple user sessions are combined, and second, how data from multiple users are combined to result in a single action series.

Foldit Variables Action series built from *Foldit* data contain 14 variables. These variables represent 28 different user actions that I grouped into 14 categories according to the what in-game tool they involve. Three of the variables reflect the use of different constraints that change how other tools will affect the protein: *bands*, which pull two parts of the protein together like a rubber band, *locks*, which freeze a portion of the protein, and *cuts*, which disconnect a part of the protein. Five variables record the use of low-level automation: *global minimization* and *local minimization* use a gradient descent process to try and improve the

[1] This approach courtesy of David Hallec, personal communication

(a) The action series for a user's multiple sessions are combined into a single series with arbitrary noise (highlighted in dashed red) in between.



(b) The action series for multiple users are then similarly combined into a single series with the same arbitrary noise (highlighted in dashed red) in between.

Figure 3.4: Part (a) shows how multiple sessions are combined into a single series. Part (b) shows how these combined series for each user are similarly combined into one series for all the input data.

solution either globally or in a specific region, *rebuild* and *build* both search a library of real-world protein fragments and attempt to replace portions of the protein with statistically realistic structure, and *repack* tries to brute-force optimize the arrangement of sidechains (chemical groups branching off the main protein backbone). Four variables relate to navigating of the problem-solving process: users can *save* their current solution and return to it later with a *load* action, and do the same with the secondary structure designations via *secondary-structure save* and *secondary-structure load*. One variable corresponds to the most frequently used secondary structure assignment, the *assign loop* action. The final variable corresponds to the *idealize* action, which sets bonds in protein backbone to theoretically ideal values.

I determined which user actions to include from among the 54 different action logged in *Foldit* by setting a cutoff for how frequently each action was used. Actions with a median usage of more than five timesteps in user series that used that action at all are included, except those that open a menu, perform a selection, or indicate a generic button press. That is, only actions directly impacting the solution are included. The median usage threshold serves to exclude actions that are too infrequent to form a signal amenable to clustering.

Foldit action series are binned by solution snapshot. The *Foldit* client uploads a snapshot of the user's current work every few minutes, so each timestep of the series represents the actions taken since the previous snapshot. Furthermore, *Foldit* action series contain only actions on a user's solution path, meaning only those performed as user went from a puzzle's initial state to their overall best solution. As it is common for *Foldit* users to explore multiple solutions to a puzzle, even simultaneously by running multiple game clients, I limit the action series to the solution path to focus it on coherent, specific behaviors. Chapter 4 investigates the exploratory structure of problem-solving behavior in *Foldit*.

Mozak Variables In *Mozak* there are only four actions that directly affect the solution: *connect the dots*, *virtual finger*, *undo*, and *delete*. Each of these is its own variable in the action series. In addition, there are two variables representing changing the appearance of different overlays and navigating the image volume prior to selecting an region to trace, respectively. The former includes changing the way consensus is shown and toggling the display of hot spots and expert comments. The goal of including these six variables is to find patterns that illuminate different rhythms or styles of tracing. *Mozak* actions series are binned into 10-minute intervals.

Starcraft II Variables Given the vast complexity of *Starcraft II* gameplay, my proof-of-concept evaluation focuses on a narrow slice of the overall problem. Specifically, I select variables for the actions series that relate to the core tension between economic and military activity. The raw data used in my evaluation contains 308 different event types (there are

unique events for each unit, building, and special ability along with those for basic commands), which I group into 10 variables. Two variables represent commands a user issues to units, *OrderEcon* for commands to economic units and *OrderMilitary* for military units. Four variables record when a user invests in new units, one for economic units, *TrainWorker*, and three for military units of increasing technology levels, *TrainT1*, *TrainT2*, and *TrainT3*. I use separate variables for different technology levels to try and incorporate some of the game context into the action series. Users will not have access to the powerful and expensive *T3* units until late in the game, and many games will end before any *T3* units are created. The remaining four variables record when a user invests in buildings, which have either an economic purpose, *BuildEcon* and *BuildSupply*, or a military purpose, *BuildStandard* and *BuildStaticDefense*. I also incorporate game context by scaling the intensity of the *Order* variables by the number of units that are given the corresponding command. *Starcraft* action series are binned into 1-second intervals.

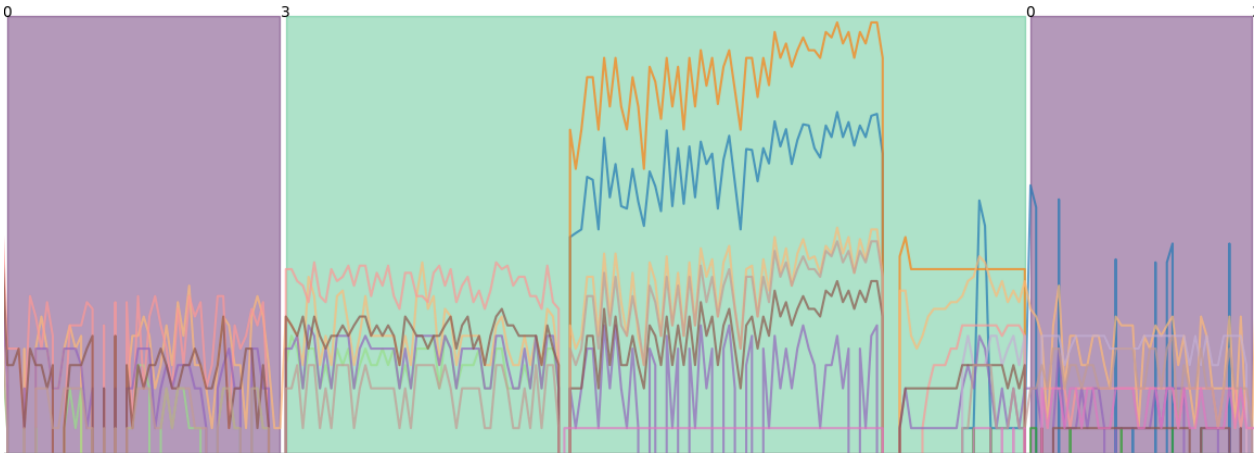
3.3.2 Recursive Clustering

The other input to the clustering step besides a single action series is a number of clusters k. Not knowing the most appropriate k a priori, I apply TICC for a range of possible k. Visually inspecting the results, however, highlights two obvious flaws with the clustering if the goal is to identify fine-grained behavior. As an example, Figure 3.5a shows an snippet of an action series for a *Foldit* user clustered using $k = 5$. The first flaw is that the middle pattern contains three clearly distinct action regimes. Ideally, all three would belong to separate clusters. The second flaw is that the first and third patterns belong to the same cluster, but appear very different. An ideal clustering would distinguish between them.

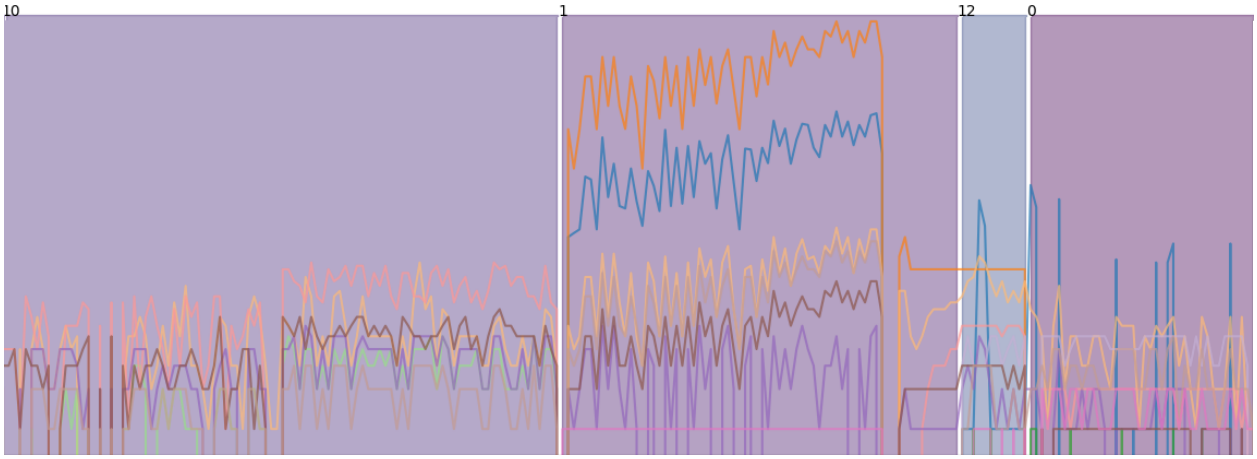
One possible avenue to getting finer-grained distinctions out of TICC might be to use a higher value of k. Figure 3.5b shows the same snippet clustered using $k = 20$. The result addresses the second flaw, but not the first. While the behavior at the beginning and end of the snippet now belong to different clusters, the second pattern still contains two distinct action regimes. In fact, increasing the value of k added a division between the second and

third pattern that may interrupt behavior better viewed as part of a single pattern. Further increasing k risks introducing many unwanted divisions.

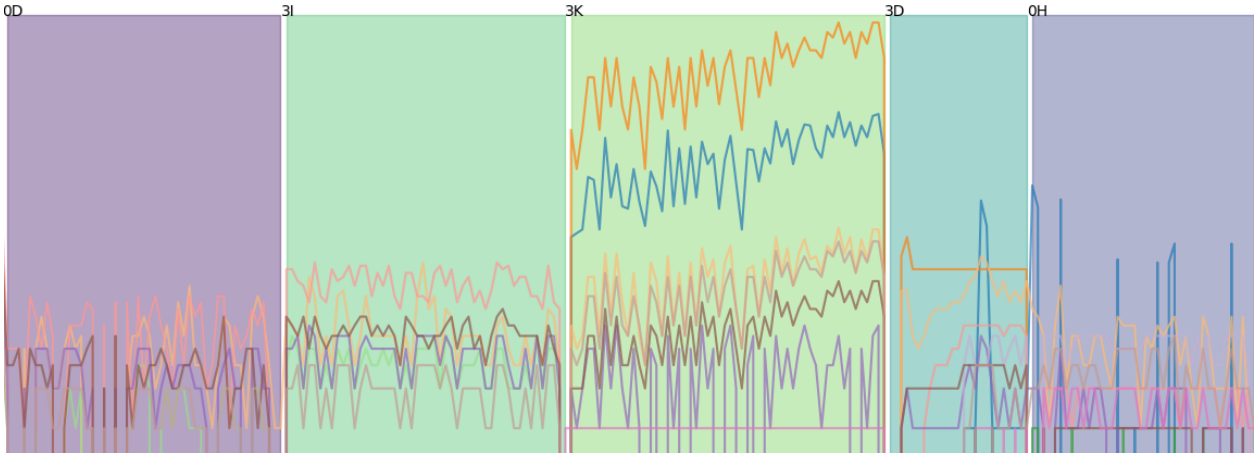
To address this difficulty, I apply the time-series clustering recursively. The key insight it to amplify the power of TICC to distinguish superficially similar behavior by recursively applying it to subsets of behavior. In particular, for each cluster c learned for a given value of k, I apply TICC to a new action series made from the instances of c. I construct this new actions series out of all the subseries belonging to c, again using arbitrary noise to connect them. This recursive clustering is again done for a range of possible k. Figure 3.5c again shows the example snippet, this time with an initial clustering of $k = 5$ and a recursive clustering of each of the two clusters in that snippet of $k = 12$. Both the flaws observed in the initial clustering have been resolved: the first and last patterns now belong to different clusters and the middle pattern has been appropriately divided along the boundaries between different action regimes. Theoretically, recursive clustering could be repeated any number of times. In this work, I apply it just once for each cluster found by the initial model.



(a) A snippet of a *Foldit* action series clustered using $k = 5$. The clustering has partitioned this snippet into three pattern instances. The first and the third instances have been placed in the same cluster (indicated by the same background color), even though they involve very different sets of actions. The middle instance contains three clearly distinct action regimes that ideally would not end up in the same pattern.



(b) The same snippet of a *Foldit* action series, now clustered using $k = 20$. This clustering successfully differentiates between the activity at the beginning and end of the snippet. Unfortunately, the second pattern instance still contains two distinct action regimes, and the division between the second and third instances interrupts what appears to be a period of consistent behavior.



(c) This version of the snippet is clustered first with $k = 5$, and then recursively with $k = 12$. Both flaws in the original clustering (a) have been corrected. The first and last pattern instances now belong to different clusters, and the three distinct action regimes in between have successfully been placed into separate instances.

Figure 3.5: A snippet of a *Foldit* action series subject to one clustering at $k = 5$ (a) and another at $k = 20$ (b). The clustering in (a) has two major flaws, only one of which is addressed by increased k. Part (c) shows how both flaws are resolved through recursive clustering.

3.3.3 Model Selection

At this point in the pattern extraction pipeline, there has been a proliferation of models: a model for each value of k in the initial clustering, and then a model for each recursive clustering of one of the initial clusters, again at varying values of k. Due to the recursive nature of the clustering, the task is not to select one single model. Rather, I must select a combination of models—one for the initial clustering and then one of the recursive models (or no recursive model if the initial cluster is sufficient) for each of those clusters. Model selection is a classic machine-learning problem, and there are many standard approaches. These include information-theoretic criteria, such as the Bayesian Information Criterion, and model accuracy on a test dataset (assuming some ground truth labeling).

Pattern extraction does not employ any of these standard methods, but instead endeavors to select the model which yields the most understandable set of patterns. The immediate goal of explaining what high and low performers do differently motivates this design choice. The quality of these explanations, as well as the ability to teach any identified high-performing patterns to low performers, relies in part on domain experts being able to interpret the extracted patterns. Hence, the *optimal* model in this context is the maximally interpretable model.

My algorithm for selecting the most understandable model relies on (1) operationalizing what information a domain expert considers when interpreting a pattern and (2) a set of criteria that use this to define an optimal model. For the former, I define *ubiquity*, a per-variable metric of how much that variable is active in a given pattern.

Definition. ***Ubiquity.*** *For an action series s of length n with a variable x, let* $\text{active}(s, x)$ *be the number of timesteps in s in which $x > 0$. Let* $\text{ubiquity}(s, x) = \frac{\text{active}(s,x)}{n}$.

Using this definition, I can compute the ubiquity of each pattern instance (i.e., each separate action subseries produced by the clustering). There are, of course, many properties of a pattern instance that ubiquity does not capture: cadence, intensity, etc. However, by focusing on which variables are present, it is a good first-order assessment of what is going

on within a particular pattern.

In terms of criteria for an optimal model, I define *dispersion*, a measure of how much a model generates patterns that contain a conflicting mix of ubiquities, and *distinctiveness*, a measure of how much a model generates non-overlapping patterns. Before defining these measure, I will define key terminology.

- **model**: a clustering of an action series for a given choice of k. A model generates a set of patterns, which I will write as $\mathbb{P}$.

- **pattern instance**: a portion of an action series labeled as a particular pattern by the clustering step. For the purposes of model selection, I view a pattern as the collection of all of its instances across the whole dataset.

- **median absolute deviation** (MAD): a measure of variability robust to outliers. It is computed by (1) finding the median of the data, (2) finding the absolute value difference between each value and this median, and (3) taking the median of these differences. I use it instead of standard deviation to reduce the influence of outlier pattern instances in the model selection process.

Definition. ***Dispersion.*** *Let P be a pattern with instances p_1, p_2, ..., p_n across the entire dataset. Let V be the set of action variables in the current domain. Let $u_i = \{\text{ubiquity}(p_i, x) | x \in V\}$. Hence, u_i is a vector representing the the ubiquity of each variable (i.e., how active that variable is) in pattern instance p_i. Let* $\text{deviation}(P)$ *be the mean of the element-wise MAD of all u_i computed from instances p_i of P. That is, for each action series variable, I compute its variation in terms of MAD across all instances of pattern P, then average these to get a single value for the deviation of pattern P. The more a pattern appears to represent a single coherent kind of behavior, the lower its deviation will be. For a model M that generates a set of patterns $\mathbb{P}$, let* $\text{dispersion}(M)$ *be the weighted mean of* $\text{deviation}(P)$ *over all $P \in \mathbb{P}$ with weights proportional to the number of instances of P (i.e., a convex combination with coefficients proportional to how often each pattern occurs).*

Definition. ***Distinctiveness.*** *Let P be a pattern with instances p_1, p_2, ..., p_n across the entire dataset. Let V be the set of action variables in the current domain. Let $u_i = \{\text{ubiquity}(p_i, x) | x \in V\}$. Hence, u_i is a vector representing the the ubiquity of each variable (i.e., how active that variable is) in pattern instance p_i. Let* $\text{mode}(P)$ *be the element-wise statistical mode of all u_i computed from instances p_i of P. That is, compute an exemplar or canonical ubiquity for pattern P by finding the mode of the ubiquity for each action series variable across all instances of pattern P. I use these canonical representations to measure the distance between patterns. For a model M that generates a set of patterns $\mathbb{P}$, let* $\text{nn-dist}(P)$ *be the minimum Manhattan distance from* $\text{mode}(P)$ *to* $\text{mode}(Q)$ *for all $Q \in \mathbb{P}$ where $Q \neq P$. In other words,* $\text{nn-dist}(P)$ *is the distance to P's nearest-neighbor using Manhattan distance on the modal ubiquity. The more distinctive a pattern appears to be, the higher this distance. Let* $\text{distinctiveness}(M)$ *be the mean* $\text{nn-dist}(P)$ *over all $P \in \mathbb{P}$.*

Minimizing the dispersion criteria selects for a model that generates coherent patterns. That is, the instances of each pattern tend involve the same action variables in similar proportions, making it potentially easier to understand what it means for a user to employ that pattern or what kind of activity that pattern represents. I employ a weighted mean to lessen the importance of patterns that occur very infrequently.

Maximizing the distinctiveness criteria selects for a model that generates patterns each corresponding to unique activity. This is desirable when trying to identify patterns associated with higher performance. The closer pattern extraction can get to a one-to-one correspondence between patterns and kinds of activity, the easier it is to use patterns to identify beneficial activity. Distinctiveness also plays a role in mitigating the bias of dispersion to select models that bifurcate user activity into many similar patterns. I use the statistical mode instead of computing a centroid because a centroid could produce a canonical representation of a pattern that is very different from any instance of that pattern.

I combine these two criteria to perform model selection as follows. First, I use dispersion to generate a set of *candidate models*, each of which comprises a choice for the initial value of

k and the subsequent recursive values of k. These choices fully determine the set of patterns produced. I generate one candidate model for each value of k in the initial clustering, selecting all the recursive ks to minimize dispersion. I then rank the candidate models by dispersion (ascending) and distinctiveness (descending). The optimal model is the candidate model with the minimum sum of its two ranks. This algorithm prioritizes minimizing dispersion (i.e., uses it to select the candidate models), as I consider the interpretability of coherent patterns critical to pattern extraction, with distinctiveness playing an important role in the final choice. In the following evaluation section, I use *optimal model* to refer to the model selected using this procedure.

3.4 Evaluation

To evaluate the capabilities of the pattern extraction technique described above, I applied it to three problem-solving domains. The most mature application (and most mature domain) is to data from *Foldit* users. I show the striking diversity of behavior among the top echelon of *Foldit* users, and conduct an analysis to identify patterns associated with higher performance. I also use the patterns produced by pattern extraction to predict user performance, comparing pattern-based models to a baseline under various conditions.

To demonstrate the generality of pattern extraction, I apply it to *Mozak*, and perform the same set of analyses. Despite *Mozak*'s relatively small user base and short history, pattern extraction is still able to produce insights into patterns of behavior that drive performance. Finally, I conduct a proof-of-concept application of pattern extraction to *Starcraft II*. Though simplistic and limited in scope, this application nonetheless shows that pattern extraction can meaningfully structure user behavior in a domain very different from either scientific-discovery game.

3.4.1 Foldit

Method

The evaluation of pattern extraction on *Foldit* used data from 10 de-novo freestyle prediction puzzles released between Spring 2016 and Spring 2017. De-novo freestyle puzzles are particularly open-ended prediction puzzles where there is no template for the protein's initial structure. For this evaluation, I used only individual solving efforts (i.e., soloists), resulting in 1,317 user sessions from 358 unique users.

To gauge the effect of the amount of training data used to learn patterns in the clustering step, I applied pattern extraction to different numbers of puzzles: one puzzle, two puzzles, three puzzles, four puzzles, five puzzles, and all 10 puzzles. In each case, I used nine different values for k in the initial clustering (5, 6, 7, 8, 9, 10, 12, 15, and 20), and four different values of k for each recursive clustering (3, 6, 9, and 12). For each subset of training data, I use the resulting pattern models to predict patterns on puzzles not included in the training data. I then select the optimal model using the patterns from all 10 puzzles. Though the size (i.e., number of patterns) of the optimal model varies widely with different training data (see Table 3.2), there was little difference in the predictive power of the resulting patterns. As detailed in the *Predicting User Performance* Section below, the pattern model trained on three puzzles had the best combination of higher predictive power and fewer total patterns to facilitate more tractable interpretation. Hence, the next section uses this model to identify high-performing behavior.

Identifying High-Performing Behavior

A key goal of pattern extraction is to help address the question *what do high and low performers do differently?* One straightforward diagnostic is to simply look at what the top-performing users are doing. That is, are there any trends in the patterns these users employ? I define the top echelon of *Foldit* users as those who have participated in more than 50 puzzles with a median performance in the top 15 percent. Figure 3.6 shows how the 30

users meeting this definition used patterns on the 10 puzzles in this evaluation.

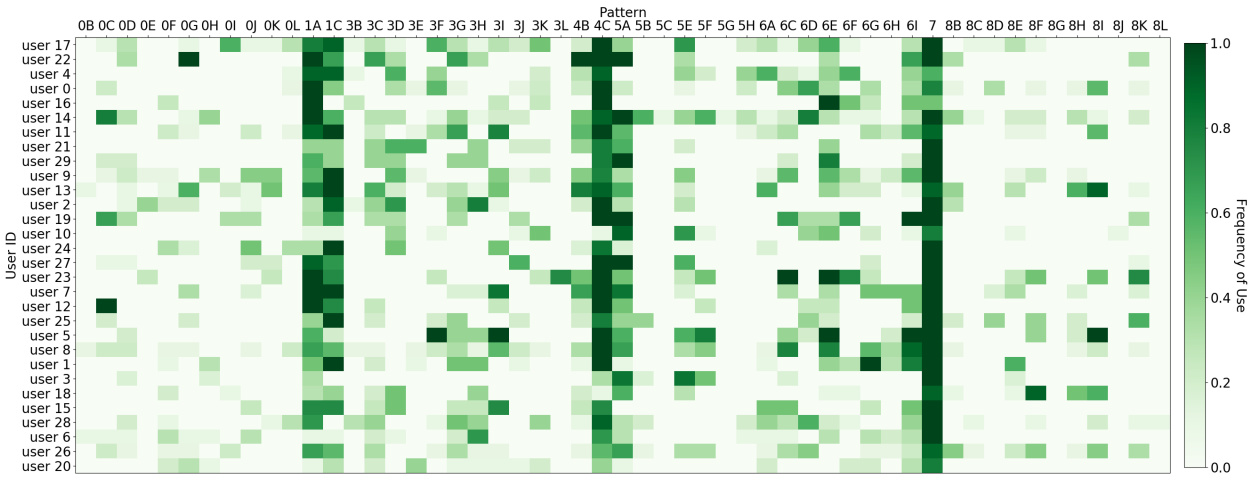


Figure 3.6: This figure shows the pattern usage of top *Foldit* users across the 10 puzzles used in this evaluation. Each row corresponds to a user who has participated in more than 50 puzzles overall with a median performance in the top 15 percent. A user ID labels each row along the left-hand side and users are sorted according to their median performance, with the highest performer at the top. Each column corresponds to one of the 53 patterns in the model. The pattern labels along the top encode how that pattern was produced by the clustering: the number indicates the cluster that pattern belonged to in the initial clustering, while the letter indicates the cluster that produced that pattern in the recursive step. Some numbers and letters are missing because the each clustering step assigns at least one cluster to the noise used to connect action series together. In the case of pattern 7, the optimal model did not include any recursive clustering. The shade of each cell indicates how frequently that user employed that pattern, with the darkest shade meaning they always used that pattern and no shade meaning they never used that pattern. This visualization shows the substantial diversity in behavior among top *Foldit* users.

As this visualization shows, there is substantial diversity among top *Foldit* users. Many patterns see only rare use (e.g., 8D), while others are heavily used, but only by a handful of top users (e.g., 8I). Others have broad, low-frequency usage, such as 0F. Even among the few patterns heavily used by most top users (e.g., 1A and 4C), there are notable outlier users who employ these popular patterns very infrequently. On the whole, this is a very surprising finding. Even considering the complexity and open-ended nature of *Foldit* as a problem-solving domain, I would have expected clear trends to emerge in the relationship

between expertise and patterns of behavior.

Instead, I find evidence for myriad paths to success in *Foldit*, emphasizing its richness as a problem-solving environment. This finding also shows the need for new techniques capable of making sense of this jungle of problem-solving behavior. Prior approaches, applied successfully to tutoring systems and educational games, often rely on a gold standard expert trace, known solution path, or small potential action space. Building deep understanding of complex and diverse open-ended domains like *Foldit* will require new data-driven analytical techniques such as pattern extraction.

Despite the diversity among high-performing *Foldit* users, there could still be patterns whose usage points to increased overall performance. I test whether each of the 53 patterns is associated with increased performance by comparing the performance of sessions where the pattern is used to sessions where it is not. Using a Mann-Whitney U test with an α of 0.000019 (an initial α of 0.001 with a Bonferroni correction for the 53 comparisons), I find 32 patterns in which users of that pattern significantly outperform non-users.

This analysis overlooks a key confounding factor, however: taking more actions leads to higher performance. In practice, when any particular pattern is used, those using it are taking more actions on average than those who are not, as using a pattern necessarily involves taking actions. Indeed, for all of the 32 patterns associated with significantly increased performance, the average user took more actions than the average non-user. An ideal apples-to-apples comparison would have users and non-users investing similar amounts of effort (i.e., taking similar numbers of actions) in order to better isolate the impact of using a particular pattern. I simulate this by increasing the performance of each pattern's non-users as if they had taken as many actions as that pattern's users. To do this mathematically, I compute a linear fit of the relationship between the log of the actions a user takes and the user's performance (see Figure 3.7). I then redo the users/non-users comparison for each pattern, using the slope of the linear fit to adjust the performance of non-users. Specifically, I take the difference between the log mean actions take by users and the log mean actions taken by non-users, and add this difference times the slope of the linear fit to the performance of every non-user. This

works to adjust the distribution of total actions taken by non-users to match the distribution of users. In this way I assess which patterns may offer performance benefits above and beyond a generic increase in actions taken.

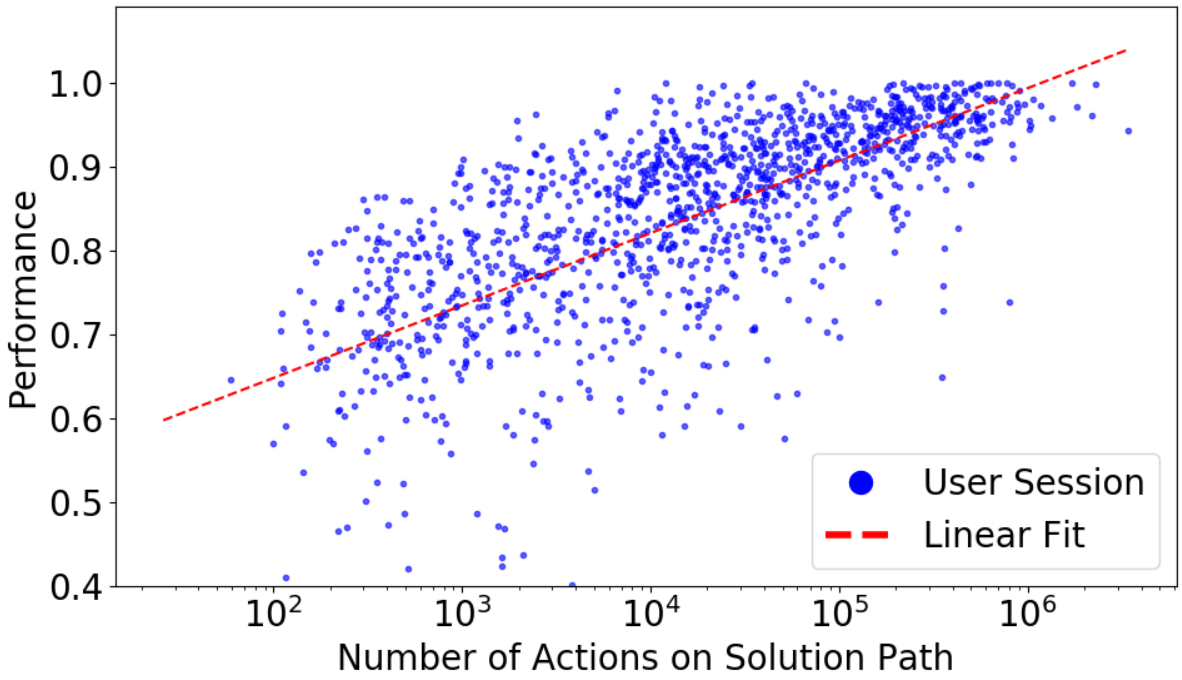


Figure 3.7: A plot showing the relationship between performance and the log of the number of actions taken on a user's solution path in *Foldit*. The number of actions is the overall total, independent of any particular pattern. I use the slope of the shown linear fit to account for additional overall actions when identifying higher-performing patterns.

Using this adjustment to account for differences in aggregate actions, there are 11 patterns where users significantly outperform non-users. Table 3.1 lists them along with number of user sessions in the evaluation dataset (out of 1,317) where the pattern was used, the rank-biserial correlation measure of effect size, and a description of the pattern. One prominent theme is that seven of these patterns are versions of a *repack-rebuild* pattern, including those with the largest effect sizes. This suggests this kind of pattern plays an important role in achieving good solutions in *Foldit*.

Pattern	User sessions	Effect Size	Description
3I	171	0.339	Repack-rebuild with standard five-variable combination: save, repack, rebuild, global minimization, and reset
3C	84	0.310	Repack-rebuild with somewhat lower use of global minimization and very high use of secondary structure tools
3D	86	0.304	Repack-rebuild with high intensity rebuild and moderate use of secondary structure tools
7	810	0.301	Grab bag of global minimization combined with other actions
1C	379	0.290	Heterogeneous set of repack-rebuild variants, generally higher intensity rebuild with occasional banding and local minimization
0G	126	0.287	Repack-rebuild with high intensity global minimization and low intensity rebuild. Also features idealize and secondary structure loading
3G	225	0.278	Repack-rebuild with high use of secondary structure tools
6E	124	0.256	Band-and-minimize using local minimization with and without global minimization
3H	104	0.253	Repack-rebuild with high use of reset and secondary structure tools
1A	480	0.216	Combination of cut and global minimization
6I	193	0.214	Band-and-minimize using local and global minimization

Table 3.1: *Foldit* patterns associated with increased performance after adjusting for the generic benefit of taking additional actions. The associations are established by a Mann-Whitney U test with an α of 0.000019 (an initial α of 0.001 with a Bonferroni correction for the 53 comparisons) comparing the performance of user sessions where a pattern was used to the performance of user sessions where it was not used. The user sessions column lists the number of sessions in which the pattern was used out of the 1,317 in the evaluation dataset. Rows are ordered by rank-biserial correlation, which I use to estimate effect size.

All repack-rebuild patterns include the variables for repack, rebuild, saving, resetting, and global minimization, and different versions include other variables as well. Since the rebuild tool tries to inject statistically realistic structure into a user's solution, and save and reset

serve to navigate the solving process, repack-rebuild patterns likely serve a key exploratory purpose by searching realistic modifications to the current solution. The use of repack and global minimization aid the exploration by continuously funneling the solution into the lowest energy state. Notably, versions of repack-rebuild that involved heavy use of banding (e.g., 0D, 0J, 0K) were not associated with increased performance, suggesting users might benefit by replacing this behavior with a more productive variation.

The other four higher-performing patterns all relate to optimization. Pattern 7 had the highest effect size of these four, and represents a grab bag of optimization behavior that combines global minimization with other actions. It is one of only two patterns occurring in a majority of user sessions, and if a user is not doing something that falls under its umbrella, they are likely just not doing sufficient optimization. Patterns 6E and 6I are *band-and-minimize* patterns that combine banding along with local and often global minimization. By contrast, other band-and-minimize patterns, and other minimization-based patterns in general, tend to also include locking. The fact that none of the higher-performing patterns include any locking suggests that locking is a less effective or hard to use constraint on optimization. Finally, pattern 1A combines the cut tool with global minimization. As it is the third-most used pattern, and one of the only patterns to involve cut, it could make a good candidate pattern for *Foldit* to try and actively promote: there is a lot of existing expertise within the community, the pattern employs a tool not seen in other contexts, and it appears to benefit its users.

Predicting User Performance

Another way to evaluate the utility of the patterns produced by pattern extraction is to use them to predict user performance. As a baseline for comparison, I fit a model of three features: number of actions taken on the solution path, median prior performance, and experience (i.e., number of puzzles the user has participated on). To evaluate the predictive power of the discovered patterns, I fit two *pattern-based models*. The first uses median prior performance, experience, and features for the number of actions taken in each pattern (referred to as

pattern-action models). The second uses all three baseline features as well as binary features for whether each pattern was used (referred to as *pattern-use models*).

For all these models, I use Gradient Boosted Regression Trees (GBRT) [58] to predict the performance of a user session based on the corresponding features. I use cross-validated recursive feature elimination (RFE) [58] to select the optimal subset of features, and score each model according to the cross-validated coefficient of determination R^2 of the prediction. The baseline model has a cross-validated R^2 of 0.642. The relative importance of the baseline model features is shown in Figure 3.8. These importances can be estimated in GBRT models by averaging the rank of each feature across all the decision trees in the ensemble.

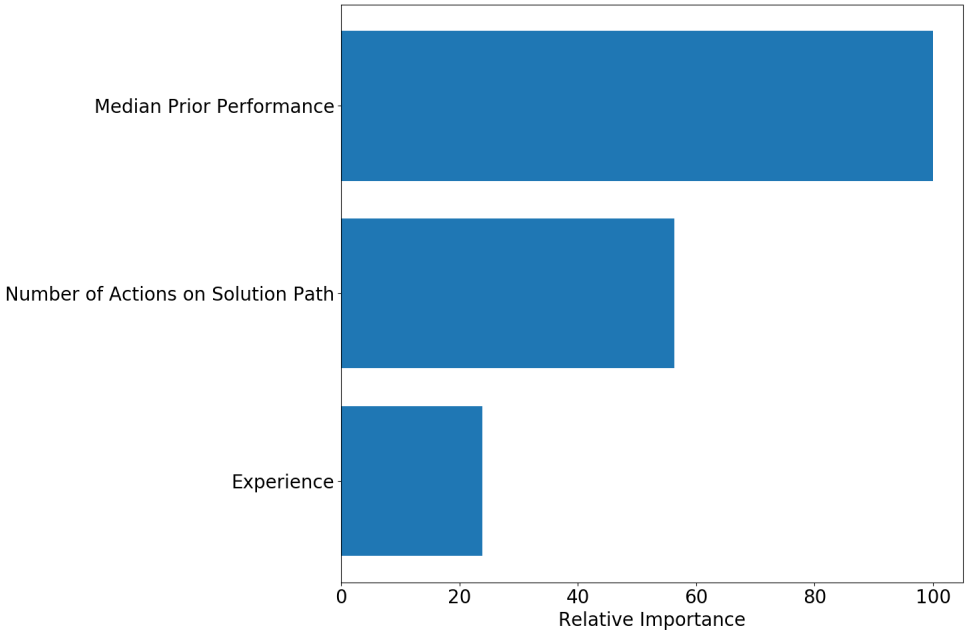


Figure 3.8: The relative feature importances for the baseline model predicting user performance in *Foldit*. The value for most important feature is set to 100, and the remaining features are scaled accordingly. Especially for a long-running environment like *Foldit*, it is no surprise that prior performance conveys a lot of information.

As discussed above, I applied pattern extraction to different subsets of the evaluation dataset to gauge the effect of the amount of training data on the patterns learned in the

clustering step. Table 3.2 gives the size of the optimal model and the pattern-based model results for each subset. All of the pattern-action models fail to outperform the baseline. All but one of the pattern-use models improve upon the baseline, though the improvement is small. The feature importances for the pattern-based models follow the same structure as the baseline model: prior performance as the most important, then the number of actions, and finally experience.

Training puzzles	Total patterns	Action features	Action R^2	Use features	Use R^2
1	104	19	0.636	5	**0.657**
2	47	13	0.621	3	0.638
3	53	50	0.626	10	**0.654**
4	69	24	0.632	23	**0.651**
5	68	19	0.630	17	**0.644**
10	29	19	0.621	20	**0.646**

Table 3.2: Optimal model sizes and R^2 values for pattern-based models predicting user performance in *Foldit*. Each row lists the results for a different number of *Foldit* puzzles used to train the TICC clustering models. While the total patterns in the optimal clustering model and the number of pattern features included in the pattern-based models (Action features and Use features) vary considerably, the R^2 values are very consistent. Those that exceed the baseline R^2 are listed in bold, but they all fall in a narrow range.

I see two plausible reasons why patterns fail to meaningfully enhance predictions of user performance relative to the baseline. First, the number of actions on the solution path turns out to be a very strong measure of effort. In a problem domain like *Foldit* that benefits from extended refinement of solutions, a good metric for user effort will go a long way towards effectively summarizing activity that leads to good performance. Nevertheless, patterns do contribute non-zero predictive power in the results, and obviously do far more to illuminate specifically what users are doing. Overall actions may be an effective measure for predicting performance, but do nothing to enable intelligent interventions by a problem-solving system. Prompting a user to *take more actions* is highly unlikely to have a useful training or corrective impact.

The predictive power of overall actions comes in part from how effectively it can separate

low-effort from high-effort users. However, the context of a given user—whether they are low-effort or high-effort, whether they are a veteran *Foldit* user or a novice—could have an important role in determining the properties of behavior most useful for predicting their performance. Hence, I evaluate the utility of pattern-based features for predicting the performance of various cohorts of users. In particular, I fit baseline and pattern-action models for eight different cohorts based on experience and overall actions. The experience cohorts are the user sessions where the user has participated on more than 10, 50, and 100 puzzles, as well as a cohort where the user has participated on at most 50 puzzles. Similarly the action cohorts are sessions where users took more than 1,000, 10,000, and 100,000 actions, as well as a cohort users took at most 10,000 actions. Table 3.3 contains the experience cohort results, and Table 3.4 contains the action cohort results.

Cohort	Sessions	Users	Baseline R^2	Action features	Action R^2
More than 10 puzzles	1,139	239	0.602	31	**0.608**
More than 50 puzzles	806	136	0.657	44	0.654
More than 100 puzzles	303	80	0.675	1	**0.676**
At most 50 puzzles	511	264	0.501	1	0.477

Table 3.3: The results for predicting the performance in *Foldit* of cohorts determined by how many puzzles users have participated in. The size of each cohort is given in user sessions and unique users. For performance predictions, the R^2 of the baseline model, the number of pattern features included in the pattern-action model, and the pattern-action model R^2 are provided (pattern-action models that outperform the baseline are bolded).

The baseline and pattern-action models have almost identical quality on the experience cohorts. It is interesting that on the more than 100 puzzles cohort, the best pattern-based model uses just prior performance, experience, and actions taken in pattern 7. The absence of other pattern features suggests highly-experienced users may predictably structure their work such that the number of actions taken optimizing in pattern 7 conveys as much information as the number of actions taken overall. The high importance of prior performance persists throughout the cohort models. A lone exception is the baseline model for the at most 50 puzzles cohort, where the number of actions is the most important feature.

Cohort	Sessions	Users	Baseline R^2	Action features	Action R^2
Above 1,000 actions	1,118	265	0.512	28	**0.527**
Above 10,000 actions	783	177	0.428	25	**0.484**
Above 100,000 actions	303	79	0.226	52	**0.279**
At most 10,000 actions	534	252	0.431	0	0.404

Table 3.4: The results for predicting the performance in *Foldit* of cohorts determined by how many actions users took. The size of each cohort is given in user sessions and unique users. For performance predictions, the R^2 of the baseline model, the number of pattern features included in the pattern-action model, and the pattern-action model R^2 are provided (pattern-action models that outperform the baseline are bolded).

The results for the action cohorts show how the predictive power of patterns relative to overall actions rises dramatically when looking at just higher-effort users. Pattern-action models outperform the baseline on all three cohorts with a minimum action threshold. Furthermore, though the model quality declines as the threshold increases, the relative advantage of pattern features grows. On above 1,000 actions, the pattern-action model R^2 is three percent higher than the baseline, but is 13 percent higher on above 10,000 actions, and 23 percent higher on above 100,000 actions. It is also noteworthy that the pattern-action model on above 100,000 actions makes more use of pattern features than any other model in the *Foldit* evaluation, with 34 different pattern features having an importance above one percent. Results on the at most 10,000 actions cohort are interesting in the opposite direction: no pattern features are included in the model at all. This highlights a potential weakness of the pattern extraction technique—it almost certainly matters what a user does even when taking at most 10,000 actions, but patterns currently have no predictive power in this context.

3.4.2 Mozak

Method

The evaluation of pattern extraction on *Mozak* used data from 90 challenges released between Spring 2018 and Spring 2019. This data contained 1,527 user sessions from 352 users. I used

the same nine values of k for the initial clustering (5, 6, 7, 8, 9, 10, 12, 15, and 20), but a more concise set of two values for k in each recursive clustering (5 and 10). The optimal model produced by pattern extraction on this data consists of 63 patterns.

To measure a user's performance on a given *Mozak* challenge, I use the number of neuron reconstruction nodes they contributed to the final consensus trace for that challenge. This is essentially a direct quantification of their impact on the scientific output. While I use the full data for training the clustering models, I use a subset of 27 challenges for identifying high-performing behavior and predicting user performance. This subset, consisting of 418 user sessions from 128 users, is those challenges for which users' contributions to consensus have been computed and exported in a format amenable to my analysis.

Identifying High-Performing Behavior

As with the *Foldit* evaluation, I examine pattern use among the top echelon of *Mozak* users and test whether any patterns are associated with increased performance over the benefits of additional generic actions. *Mozak* is a much younger project, so I define of the top echelon as users who have participated in more than five challenges (instead of 50) with a median performance in the top 15 percent. Figure 3.9 shows how the eight users meeting this definition used patterns on the 27 challenges in this evaluation.

Two key aspects of the *Mozak* user base greatly influence the results of this evaluation: (1) the consistent user base is relatively small and (2) overall contributions are dominated by two *super users* (users 4 and 1) who contribute an order of magnitude more than all other users. Nevertheless, Figure 3.9 shows diversity among top users. Users vary in the emphasis they put on different styles of connect-the-dots-based tracing (patterns 3G, 4B–4I), while others prefer to rely on the virtual finger tool (patterns 3F, 3H, 3I, 5D). A majority of top users clearly favor connect the dots. The two super users look similar, including frequent use of the pattern representing idle time, pattern 0, suggesting more pausing and resuming work instead of doing everything in one sitting.

To test whether any patterns are associated with increased performance, I compare the

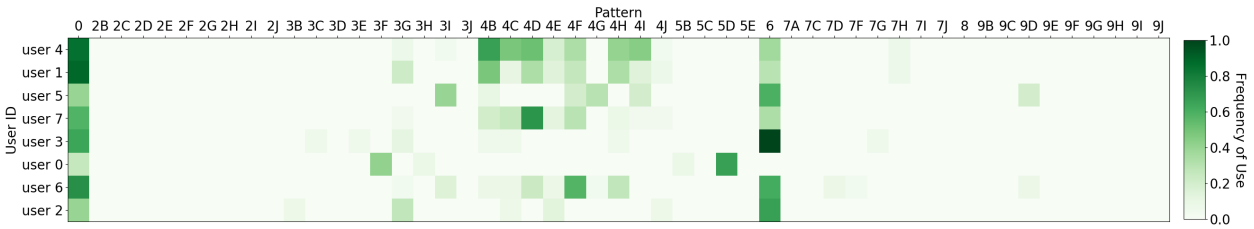


Figure 3.9: This figure shows the pattern usage of top *Mozak* users across the 27 challenges used in this evaluation. Each row corresponds to a user who has participated in more than 5 challenges overall with a median performance in the top 15 percent. A user ID labels each row along the left-hand side and users are sorted according to their median performance, with the highest performer at the top. Each column corresponds to one of the 63 patterns in the model, with some rarely-used patterns omitted to make the visualization easier to understand. The pattern labels along the top encode how that pattern was produced by the clustering: the number indicates the cluster that pattern belonged to in the initial clustering, while the letter indicates the cluster that produced that pattern in the recursive step. Some numbers and letters are missing because the each clustering step assigns at least one cluster to the noise used to connect action series together. In the case of patterns 0, 6, and 8, the optimal model did not include any recursive clustering. The shade of each cell indicates how frequently that user employed that pattern, with the darkest shade meaning they always used that pattern and no shade meaning they never used that pattern. This visualization shows the diversity in behavior among top *Mozak* users, while also showing the similarity between the two *Mozak* super users, 4 and 1.

contributions to consensus for sessions where a pattern was used to the contributions for sessions where it was not. Using a Mann-Whitney U test with an α of 0.000016 (an initial α of 0.001 with a Bonferroni correction for the 63 comparisons) I find eight patterns where users show significantly higher performance. As in *Foldit*, however, taking additional actions has a strong relationship with increased performance in *Mozak*. Thus, I use a linear fit of the log actions taken versus the log contribution (see Figure 3.10) to adjust the performance of non-users. Specifically, I take the difference between the log mean actions taken by users and the log mean actions taken by non-users, multiply this by the slope of the linear fit, and add it to the log of the contribution of each non-user.

With this adjustment, I find two patterns associated with increased performance: 4B and 4D with effect sizes of 0.467 and 0.458, respectively. This finding deserves cautious interpre-

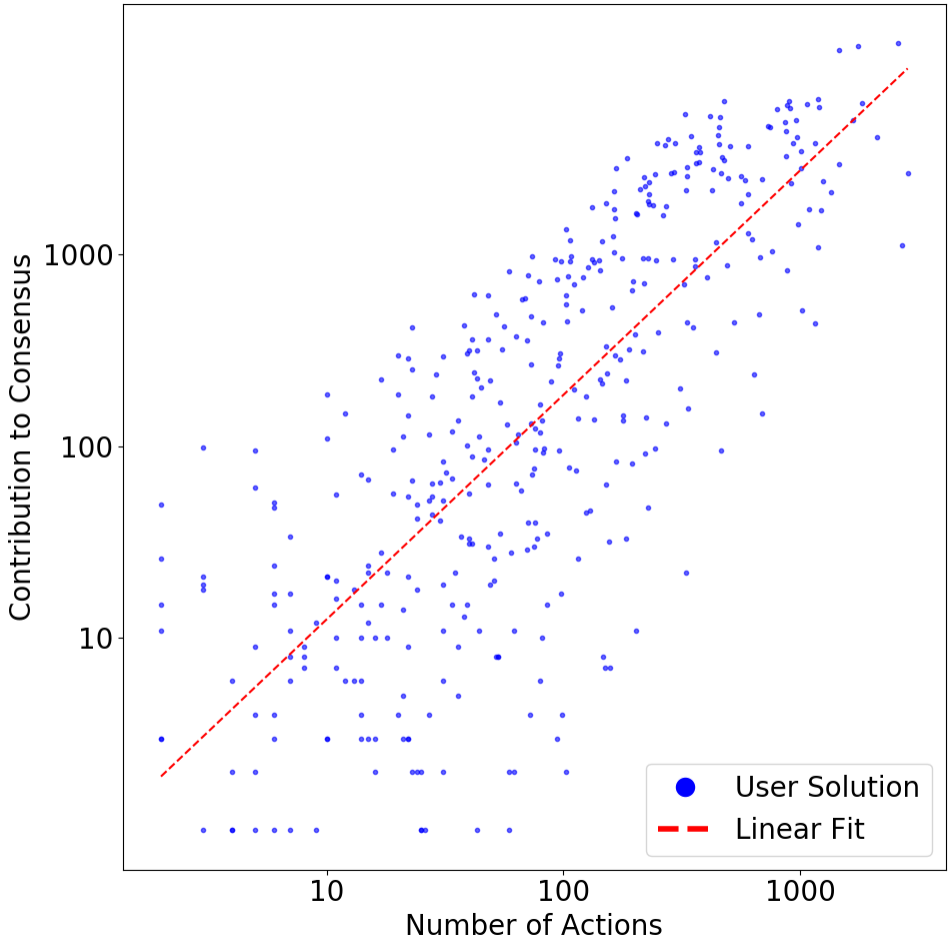


Figure 3.10: A plot showing the relationship between log number of actions taken and log contribution to consensus in *Mozak*. The number of actions is the overall total, independent of any particular pattern. I use the slope of the shown linear fit to account for additional overall actions when identifying higher-performing patterns.

tation. These two patterns are favored by *Mozak*'s two super users, and this undoubtedly makes them look more productive. Still, the specifics of 4B in particular merit discussion. Combining extended use of connect the dots with frequent, low-intensity undo and delete, 4B resembles sustained *careful tracing*. In this pattern, the user both generates substantial new reconstruction while swiftly moving to correct mistakes. Users of each of four variations

of this pattern significantly outperform non-users without any adjustment for extra actions taken. Pattern 4D, by contrast, is a *burst tracing* pattern—shorter duration uses of connect the dots with little-to-no undo or delete.

Predicting User Performance

I also evaluate the predictive power of the patterns extracted from *Mozak* data. Like the *Foldit* evaluation, I fit a baseline model of three features (prior performance, experience, and total actions taken), as well as a pattern-action model of prior performance, experience, and features for the actions taken in each pattern and a pattern-use model of prior performance, experience, total actions taken, and features for whether each pattern was used. Also as before, I use cross-validated RFE to select the optimal feature set for a GBRT model. The pattern-action model again underperforms the baseline (R^2 of 0.721 using 14 pattern-based features versus 0.747 for the baseline), while the pattern-use model improves slightly on the baseline (R^2 of 0.756 using two pattern-based features).

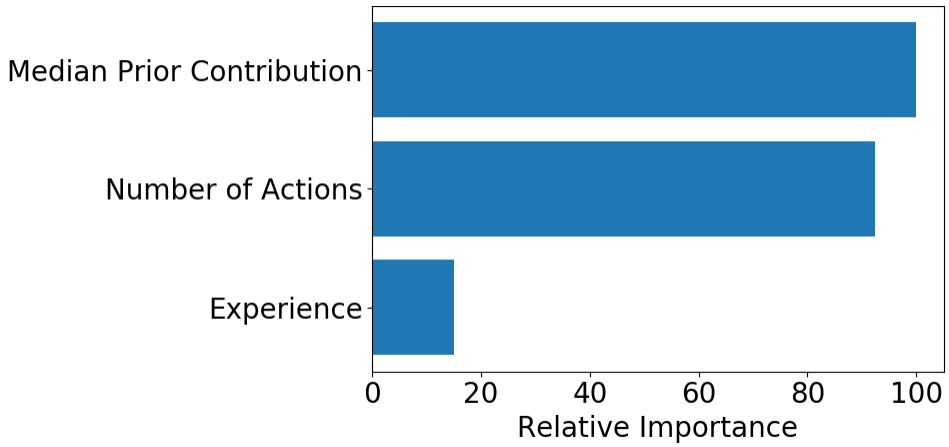


Figure 3.11: The relative feature importances for the baseline model predicting user performance in *Mozak*. The value for most important feature is set to 100, and the remaining features are scaled accordingly. Unlike *Foldit*, prior contribution and number of actions are almost equally important.

In terms of feature importance, the total actions taken are more important in *Mozak* than in *Foldit*. As Figure 3.11 shows, the number of actions are almost as important as prior performance in the baseline model's predictions. In fact, the number of actions is the most important feature for the pattern-use model. There are two explanations for why the number of actions would have greater importance for predicting performance in *Mozak*: (1) the vastly shorter user history in *Mozak* makes prior performance less reliable, increasing the relative utility of the number of actions, and (2) neuron tracing is a far more linear task than protein folding, with more reliable performance gains from additional actions. I believe both of these factors are likely at play.

Cohort	Sessions	Users	Baseline R^2	Action features	Action R^2
More than 1 challenge	237	41	0.734	12	0.614
More than 5 challenges	135	13	0.649	7	**0.697**
More than 10 challenges	81	9	0.739	8	0.690
At most 5 challenges	283	128	0.710	50	0.638

Table 3.5: The results for predicting the performance in *Mozak* of cohorts determined by how many challenges users have participated in. The size of each cohort is given in user sessions and unique users. For performance predictions, the R^2 of the baseline model, the number of pattern features included in the pattern-action model, and the pattern-action model R^2 are provided (pattern-action models that outperform the baseline are bolded). Experience itself ceased to be predictive for some of these cohorts. Cross-validated recursive feature elimination found the best-scoring baseline model for each cohort except more than 10 challenges did not include a feature for experience.

Predicting the performance of various cohorts of *Foldit* users reveals contexts where the predictive power of pattern-based features increases substantially compared to total actions taken. The same does not hold true for *Mozak*. For experience cohorts of more than one challenge, more than 10 challenges, and at most 5 challenges, pattern-action models lag behind baseline models (see Table 3.5). The same is true for the four action cohorts of above 50 actions, above 100 actions, above 500 actions, and at most 50 actions (see Table 3.6). The sole exception is the cohort where users have participated in more than five challenges.

The experience cohorts highlight the shallowness of the current *Mozak* user base. Of

Cohort	Sessions	Users	Baseline R^2	Action features	Action R^2
Above 50 actions	240	67	0.722	38	0.683
Above 100 actions	178	41	0.649	12	0.607
Above 500 actions	52	14	0.517	63	0.396
At most 50 actions	178	97	0.123	2	0.025

Table 3.6: The results for predicting the performance in *Mozak* of cohorts determined by how many actions users took. The size of each cohort is given in user sessions and unique users. For performance predictions, the R^2 of the baseline model, the number of pattern features included in the pattern-action model, and the pattern-action model R^2 are provided.

the 128 users in the evaluation data, only 41 have participated in more than one challenge, and only nine have participated in more than 10 challenges. The anomalous results on the more than five challenges cohort are likely noise due to the small amount of data in the higher-experience cohorts. It is possible there is simply insufficient data for pattern-based models to perform well on *Mozak*. The way patterns bifurcate information about user activity might require more data to be competitive with a concise baseline model. Finally, it is worth emphasizing that two super users will heavily distort a dataset of this size. These two users take an order of magnitude more actions and contribute an order of magnitude more to consensus than other top *Mozak* users, let alone the user base at large.

3.4.3 Starcraft II

Method

To perform a proof-of-concept evaluation of pattern extraction on *Starcraft II*, I collected 50 publicly available replays of online one-vs-one *Starcraft II* games from the web site `gggreplays.com`. A *Starcraft* replay contains a record of all the events that occurred during that game, which I then process into an action series for each user. Altogether, these replays represent 20.1 hours of gameplay across 100 user sessions from 62 unique users. The primary purpose of this evaluation is to demonstrate a successful minimal extension of pattern extraction to a problem-solving domain very different from scientific-discovery puzzle games

For the initial clustering, I used the same range of values for k as the evaluations on *Foldit* and *Mozak*: 5, 6, 7, 9, 10, 12, 15, and 20. As this evaluation represents a preliminary application of pattern extraction to a limited *Starcraft II* dataset, I did not perform any recursive clustering. The optimal model contains five patterns.

Economic vs Military Focus

My choice of action series variables aimed to examine the tension between economic and military activity in *Starcraft*. The five patterns in the optimal model break down into two predominantly economic patterns and two predominantly military patterns (with one rarely-used outlier pattern):

Economic Pattern 1: a mix of economic and military activity characterized by a steady stream of investments in workers, military units, and structures. Users issue orders coordinating military and economic units in roughly equal proportions.

Economic Pattern 2: a heavily economic pattern with a high volume of economic orders and very few military orders. Users make major investments in workers, economic buildings, supply buildings, and military units, suggesting passive *building-up* behavior.

Military Pattern 1: an earlier-game military pattern with frequent bursts of military orders and big investments in low-tech military units. While the sparsity of economic orders indicates the user's attention is not on coordinating economic activity, they still tend to make investments in workers.

Military Pattern 2: a later-game military pattern with extended periods of issuing military orders (e.g., engaging in running battles) and training higher-tech military units. The user's attention seems to be solely on military activity as this pattern contains the fewest economic orders and investments in workers.

Figure 3.12 shows how users progress through these patterns over the course of the games in the evaluation dataset, grouped according to the race the user chose. Each bar shows the patterns for one user in one game, with the start of the game on the left and the end of the game at the right end of the bar. Every user begins each game in one of the economic

patterns. This is a promising sign, as the first few minutes of a *Starcraft* game necessarily consists of almost entirely economic activity as users focus on building up their economies before starting to invest in a military. Users, whether on their own initiative or in reaction to their opponent, may extend this economic focus or transition to military operations early (often referred to as *boom* and *rush* strategies, respectively). Thus, it also makes sense that the patterns show users transitioning from an economic to a military pattern at different points during the game.

The most obviously flaw with these preliminary results is that users playing the Terran and Protoss races never leave the economic patterns. This is likely due to the different dynamics of the three races. While the Zerg gain the ability to train many workers in parallel as the game progresses, Terran and Protoss must produce them one at a time. With the data from all three races together, the economic activity of Terran and Protoss users throughout the game may most resemble the initial economic activity of Zerg users. Additional data might enable pattern extraction to correctly differentiate these, though learning separate patterns for each race could ultimately be necessary. In general, my current transformation of in-game events to action series is fairly simplistic and extending it to go beyond the straightforward dynamics captured here is the subject of future work.

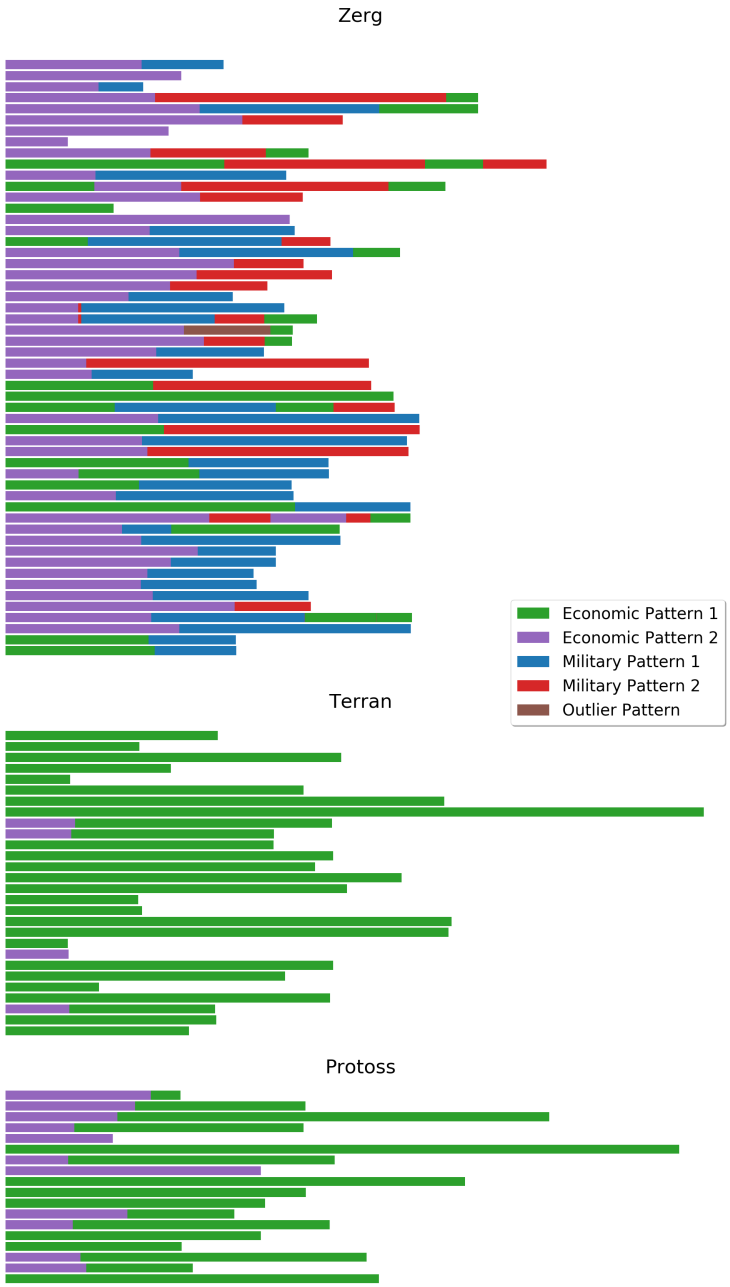


Figure 3.12: A visualization of how users progress through patterns over the course of games in the *Starcraft II* evaluation dataset. Each bar shows the patterns for one user in one game. The bars are grouped according to the race the user chose. Time runs from left to right, so the color of the leftmost part of a bar indicates the pattern a user started in and so on. The length of the bar corresponds to the length of the game.

3.5 Discussion and Future Work

My evaluation of pattern extraction demonstrates its generality and utility as an analytic approach. I successfully apply my technique to two scientific-discovery games with very different tasks, and use the resulting patterns for the same set of analyses to illuminate user behavior. I further show generality with a proof-of-concept evaluation on *Starcraft II*, a challenging domain very different from the other domains I target.

These evaluations illustrate that pattern extraction can automatically discover rich patterns of problem-solving behavior. A key feature of the pattern extraction architecture is that it limits necessary domain expertise to two key points: the design of the action series variables and the interpretation of the resulting patterns. On the other hand, it is important that the analytic framework not exclude input from domain experts. Prior work shows expert users tend to employ efficient, domain-specific strategies [55, 9, 36], and correctly identifying and diagnosing these is very unlikely without any domain expertise.

The analysis I perform on *Foldit* and *Mozak* demonstrates the power of pattern extraction to produce insights into the behavior that sets high performers apart. I identify patterns associated with higher performance significantly above that provided by generic additional user effort. In *Mozak*, the frequent use of a particular *careful tracing* pattern by the game's two super users suggests that this might be a behavior worth incentivizing or scaffolding. In *Foldit*, seven of the variants of the *repack-rebuild* pattern are associated with higher performance, prompting interesting questions for further investigation: do these different variations play different roles? Is their use situational? Should users combine multiple variants? Pattern extraction provides a foundation to identify and address these kind of questions.

Diversity among the top echelons of *Foldit* and *Mozak* users is both exciting and challenging. It demonstrates these scientific-discovery games present complex tasks with multiple paths to success. Furthermore, the lack of convergence highlights the potential for a problem-solving system to facilitate impactful cross-pollination and guide intermediate users to nearby more successful behaviors. At the same time, this diversity challenges the ability of patterns-as-

features to contribute predictive power, as few patterns see widespread usage. I also observe that no predictive power is gained on *Foldit* data by increasing the amount of training data available for the clustering step. It is possible that one *Foldit* puzzle is already a rich and fairly comprehensive dataset on user behavior, thus limiting the utility of additional puzzles.

The degree of variation in optimal model size with different clustering training data is concerning, however. It suggests that the clustering and/or model selection steps may be oversensitive to small differences in training data. In general, the model selection procedure described here is speculative. I designed it to align as closely as possible with the overall goals of pattern extraction, but I do not evaluate it in isolation. Hence, selecting the most understandable model is a design principle rather than an established property of pattern extraction.

There are numerous possible extensions to the current pattern extraction architecture that might address these concerns. As my analysis of *Foldit* and *Mozak* shows, many patterns are variations on the same theme, and some may be similar enough to be viewed as the same pattern. In this work, these relationships are discovered by manual inspection of the patterns, but much of this could be performed automatically. Using ubiquity and other metrics, pattern extraction could suggest which patterns are variations of one another or combine sufficiently similar ones. One source of information about the discovered patterns I do not use in this work are the Markov Random Fields (MRFs) the TICC algorithm learns to represent cluster structure. Though ubiquity proved a more reliable *big picture* metric for the kind of behavior involved in each pattern, the MRFs could likely enhance this picture. Finally, the pattern extraction pipeline could be made more sophisticated and adaptive. Currently, a single round of recursive clustering is applied to each initial cluster. Instead, properties like those used in model selection could guide where and to what extent recursive clustering is done. This, along with automatic combining of very similar patterns, would make the pattern extraction process more robust and accurate.

The construction of action series is another promising area for extending pattern extraction. There are many types of meaningful signals the action series as currently assembled do

not include. For example, *Foldit* action series exclude very infrequent or one-off actions such as triggering a recipe or using certain high-level manual tools. Actions series could incorporate these as a *synthetic signal* that extends their presence beyond a single-timestep spike. Alternatively, a post-clustering step could augment, annotate, or partition patterns according to the occurrence of these events. Context is another class of potential signals currently omitted from pattern extraction. This could take the form of variables for user score, a measure of change in solution state, available resources, and so on. In addition to user context, problem context could also prove valuable. Action series could include variables for what part of the problem the user is working on or unique features of the particular problem instance. These non-action variables might enable patterns to reflect the optimal context in which to use them.

Lastly, all the predictive models in my evaluation concerned users' overall performance and what patterns they used to achieve it. This ignores, however, the sequencing and timing of patterns, which may play a key role in determining their effectiveness. Furthermore, interventions by a problem-solving system will require models that predict how a user would perform *if* they took a certain approach. Modeling these situations (pattern-based sequence models for the former, predictions based on hypothetical user behavior for the latter) is an important direction for future applications of pattern extraction.

Chapter 4

VISUALIZING PROBLEM-SOLVING STRUCTURE

This chapter is based on work the author published at EDM 2017.

4.1 Introduction

As efforts in scalable online education expand, interest continues to increase in moving beyond small, highly constrained tasks, such as multiple choice or short answer questions, and incorporating creative, open-ended activities [24, 38]. Existing research supports this move, showing that problem-based learning can enhance students' problem-solving and metacognitive skills [30]. Scaling such activities poses significant challenges, however, in terms of both assessment and feedback. It will be vital to devise scalable techniques not only to assess students' final products, but also to understand their progress through complex and heterogeneous problem-solving spaces. These techniques will apply to a broad range of education settings, from purely online programs like Udacity's Nanodegrees to more traditional settings where new standards like the Common Core emphasize strategic problem solving.

A growing body of work has found that educational and serious games are fertile ground for assessing students' capabilities and problem-solving skills [21, 28]. Chapter 3 presented an automated technique for extracting patterns of user behavior from time series of user actions, demonstrating its effectiveness on two scientific-discovery games. While highly general, action time series do not leverage any of the rich structural context inherent to problem solving in one of these games, the protein-folding puzzle game *Foldit*. Exploring different approaches, backtracking to previous solutions, and navigating the highly discontinuous solutions space are all potentially crucial aspects of *Foldit* problem solving, and open-ended design problems more

generally, that *pattern extraction* does not address. In this chapter, we devise a visualization of problem-solving structure in *Foldit* and use it to identify patterns of problem-solving behavior.

Though the structural context of *Foldit* offers an important window into the problem-solving process, the size and complexity of *Foldit*'s problem space presents a corresponding challenge. Even though the logs of user interactions consist only of regular snapshots of a user's current solution (along with attendant metadata), the record of a single user's performance on a given problem frequently consists of thousands of such snapshots (which in turn are just a sparse sampling of the actual solving process). Furthermore, the nature of the solution state, the configuration of hundreds of components in continuous three-dimensional space, renders collapsing the state space by directly comparing solution states impractical. Compounding the size of the problem space is the complexity of the actions available to *Foldit* users. In addition to manual manipulation of the protein configuration, users can invoke various low-level automated optimization routines (some of which run until the user terminates them) and place different kinds of constraints on the protein configuration (*rubber bands* in *Foldit* parlance) that restrict its modification in a variety of ways. Users can also deploy many of these tools programmatically via Lua scripts called *recipes*. Taken together these challenges of ill-structuredness, size, and complexity threaten to make analysis of high-level problem-solving structure in *Foldit* intractable.

To overcome these obstacles, we devise a visualization-based methodology capable of producing tractable representations of *Foldit* users' problem-solving structure while maintaining the key encodings necessary for analysis of high-level patterns. A process of iterative summarization forms the core of this methodology, and ensures that the transformations applied to the raw data do not elide structures potentially relevant to understanding users' unique problem-solving patterns. Using this methodology, we examine user activity logs from 11 *Foldit* puzzles, representing 970 distinct users and nearly 3 million solution snapshots. Leveraging metadata present in the solution snapshots, we represent solving behavior as a tree, and apply our methodology to visualize a summarized tree showing where they branched off

to investigate multiple hypotheses, how they employed some of the automated tools available to them, and other salient problem-solving behavior. We use these depictions to determine key distinguishing features of this exploration process. We subsequently use these features to better understand the patterns of expert-level problem solving.

Our work focuses on the following research questions: (1) how can we visually represent an open-ended exploration towards a high-quality solution in a large, ill-structured problem space? (2) what are the key patterns of problem-solving behavior exhibited by individuals?, and (3) what are the key differences along these patterns between high-performing and lower-performing users in an open-ended domain like *Foldit*? In addressing these questions we find that high-performing users explore the solution space more broadly. In particular, they pursue more hypotheses and actively avoid getting stuck in local minima. We also found that both high- and lower-performing users have similar proportion of manual and automated tool actions, indicating that better performance on open-ended challenges stems from the quality of the action intermixing rather than aggregate quantity.

4.2 Methodology

Prior work has demonstrated the power of visualization to support understanding of problem-solving behavior (e.g., [35]). Hence, we devise a methodology capable of producing concise, meaning-rich visualizations of the problem-solving process in *Foldit*, and then leverage these visualizations to identify key patterns of user behavior. We are specifically interested in how users navigate from a puzzle's start state to a high-quality solution, what states they pass through in between, and what other avenues they explore.

The scale of the *Foldit* data necessitates significant transformation of the raw data in order to render concise visualizations. Without any transformation, meaningful patterns are overwhelmed by sparse, repetitive data and would be far more challenging to identify. While there are many existing techniques for large-scale tree visualization, we find clear benefits to developing a visualization tailored to the *Foldit* domain. Specifically, preserving the semantics of our visual encoding is crucial for allowing us to connect patterns in the visualization to

concrete problem-solving structure in *Foldit*. To accomplish this, the process by which concise visualization are constructed must be carefully designed to maintain these links. Hence, we devise a design methodology focused on *iterative summarization*.

This process begins by visualizing the raw data. This is followed by iteratively building and refining a set of transformations to summarize the raw data while preserving meaning. The design of these transformations should be guided by frequently occurring structures. That is, those structures that the transformations can condense without eliding pieces corresponding to unique patterns. In parallel to this iterative design, a set of visual encodings are developed to represent the solving process as richly as possible. Key to this entire process is frequent consultation with domain experts, in our case experts on *Foldit* and its community. By applying this iterative methodology for several cycles, we designed a domain-specific visualization that we use to identify structural patterns of behavior among *Foldit* users.

4.2.1 Data

For our analysis, we selected 11 prediction puzzles spanning the range of time for which the necessary data is available. Though *Foldit* has been in continuous use since 2010, the data necessary to track a user's progress through the problem space has only been collected since mid-2015. Our chosen dataset represents 970 unique users and nearly 3 million solution snapshots. These 11 puzzles are just a small subset of the available *Foldit* data. We chose a subset of similar puzzles (i.e., a subtype of relatively less complex prediction puzzles) in order to make common solving-behavior patterns easier to identify. The size of the subset was also guided by practical constraints, as each puzzle constitutes a large amount of data (20-60 GB for the data from all players on a single puzzle).

The data logged by *Foldit* primarily consists of snapshots of user solutions as they play, stored as text files using the Protein Data Bank (pdb) format. These snapshots include the current protein pose, a timestamp, the solution's score, the number of times the user has invoked each action and recipe, and a record of the intermediate states that led up to the solution at the time of the snapshot. This record, or *solution history*, is a list of unique

identifiers each corresponding to a previous solution state. This list is extended every time the user undoes an action or reloads a previous solution. Hence, by comparing the histories of two snapshots from the same user, we can answer questions about their relationship (e.g., does one snapshot represent the predecessor of another; where did two related snapshots diverge). The key relationship for the purposes of this analysis is the direct parent-child relationship, which we use to generate trees that represent a user's solving process.

4.2.2 Visualizing Solution Trees

We applied our methodology to our chosen subset of *Foldit* data to design a visualization of an individual's problem-solving process as a *solution tree*. Several key principles guided this design. First, since our goal is to discover key patterns, the visualization needs to highlight distinctly different problem-solving structures. These differences cannot be buried amidst enormous structures, nor destroyed by graph transformations. Second, the visualization must depict the closeness of each step to the ultimate solution in both time and quality to give a sense of the user's progression. Third, the user's use of automation in the form of recipes should be apparent since the use of automation is an important part of *Foldit*.

The fundamental organization of the visualization is that each node corresponds to a solution state encountered while solving. Using the solution history present in the logged snapshots of user solutions, we establish parent-child relationships between solutions. If solution β is a child of solution α, it indicates that β was generated when the user performed actions on α. One crucial limitation, however, is that a snapshot of the user's current solution is captured far less often (only once every two minutes) than the user takes actions. This means that our data is sparsely distributed along a solution's history going back to the puzzle's starting state. Hence, when naively constructing the tree from the logged solution histories, it ends up dominated by vast quantities of nodes with no associated data.

We address this issue by performing summarization on the solution trees, condensing them into concise representations amenable to analysis for important features. This summarization takes place in two stages. The first stage trims out nodes that (1) do not have corresponding

data and (2) have zero children. This eliminates large numbers of leaf nodes that we are unable to reason about given that we lack the corresponding data. This stage also combines sequences of nodes each with only one child into a single node. For the median tree, this stage reduced the number of nodes by an order of magnitude from over 12,000 nodes to about 1,600.

The second stage consists of four phases, each informed by our observations of common patterns in trees produced by the first stage that would benefit from summarization. The first phase, called *prune*, focuses on simplifying uninteresting branches. We observed many of the branches preserved by the first stage were small, with at most three children, and only continued the tree from one of those children. Prune removes the leaf children of these branches from the tree. *Collapse*, the second phase, transforms each of the sequences of single-child nodes left behind after prune into single nodes. The third phase, *condense*, targets another common pattern where a sequence of branches feed into each other, with a child of each branch the parent of the next branch. These sequences are summarized into a single node labeled CASCADE along with the depth (number of branches) and width (average branching factor) of the summarized branches. See Figure 4.1 for an example of the features summarized by these three phases. The final phase, *clean*, targets the ubiquitous empty nodes (i.e., nodes for which we lack associated data) shown in black in Figure 4.1. We eliminate them by merging them with their parent node, doing so repeatedly until they all have been merged into nodes that contain data. In addition to making the trees more concise, this step allows us to reason more fully over the trees since all nodes are guaranteed to contain data. This second stage of summarization further reduced the number of nodes in the median tree by another order of magnitude to about 300 nodes. Summarization similarly reduces the space required to store the data by two orders of magnitude. See Figure 4.2 for an example of a relatively small, fully summarized graph.

Child-parent relationships are not the only part of the data we visually encoded in the solution trees. Nodes are colored on a continuous gradient from red to blue according to the score of the solution represented by that node (red is low-scoring, blue is high-scoring). The

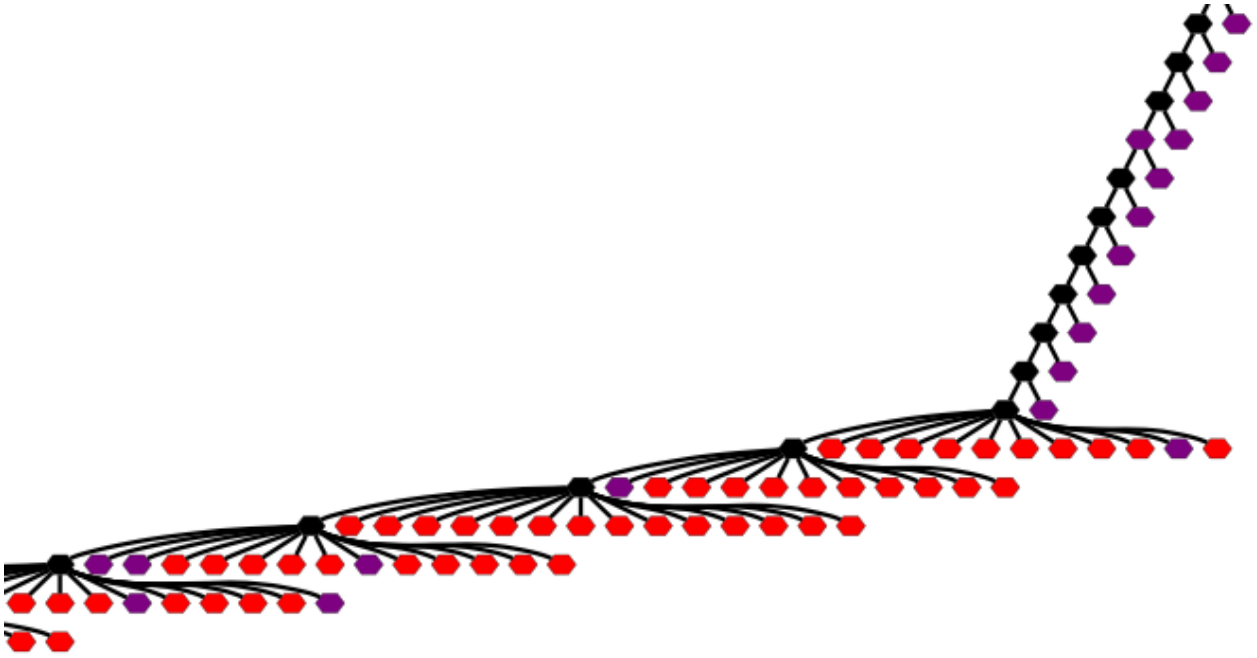


Figure 4.1: A solution tree after only the first stage of summarization. The non-black node color represents the score of the solution at that node (red is worse). The black nodes are empty in that we do not have solution data corresponding to that node. This figure also shows examples of the features targeted by the second summarization stage: *prune* and *collapse* eliminate long chains like the one on the right, and *condense* combines sequences of branches like those going down to left in single CASCADE nodes.

best-scoring node is highlighted as a yellow star. Edges are colored on a continuous gradient from light to dark green according to the time the corresponding transition took place, and the children of each node are arranged left to right in chronological order. Finally, use of automation via recipes is an important aspect of problem-solving in *Foldit*. Since the logged solution snapshots contain a record of which recipes have been used at that point, we can use this to annotate nodes where a recipe was triggered. The annotations consist of the id of that recipe (a 4 to 6 digit number) and the number of times it was started.

One major weakness in the data available to us is the lack of a consistent way to determine

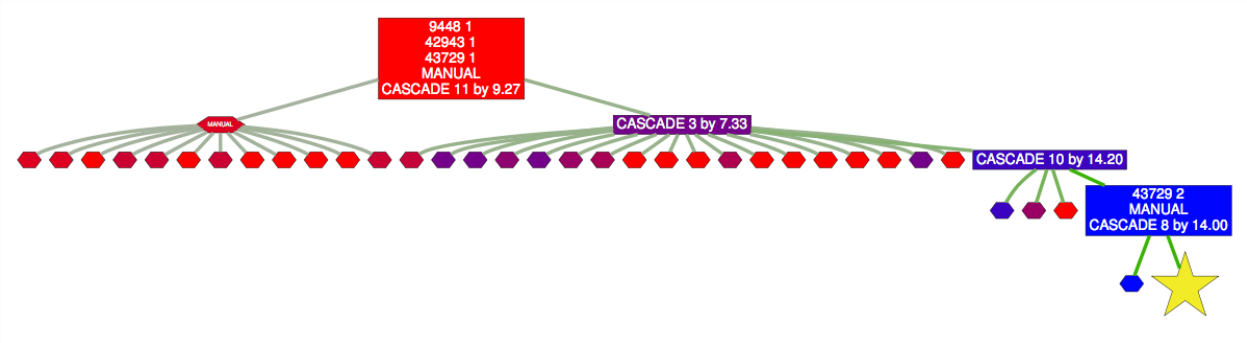


Figure 4.2: A small, fully summarized solution tree.

when the execution of a recipe ended (some recipes save and restore, possibly being responsible for multiple nodes in the graph beyond where they were triggered). We partially address this by further annotating a node with the label MANUAL whenever the user took a manual-only action at that node. This indicates that no previously triggered recipe continued past that node because no recipe could have performed the manual-only action. Since nodes in the summarized trees can represent many individual steps, it is possible for them to have several of these recipe and manual action annotations.

4.3 Results

Using visualized solution trees for a large set of users across our sample of 11 puzzles, we identify a set of six prominent patterns in users' problem-solving structure. These patterns do not encompass all solving structure in *Foldit*, but instead capture key instances in three categories: exploration, optimization, and human-computer collaboration. Future work is needed to generate a comprehensive survey of the structural patterns in these and other categories. In this analysis, our focus is on identifying a small, diverse set of commonly occurring patterns to both provide insight into problem-solving behavior, and to demonstrate the potential of our approach. In addition to identification, we also perform a quantitative comparison of how these patterns are employed by high-performing and lower-performing users to gain an understanding of how these patterns contribute to success in an open-end

environment like *Foldit*.

4.3.1 Patterns in Problem-Solving Structure

Exploration *Foldit* users are confronted with a highly discontinuous solution space with many local optima, creating a trade-off between narrowly focusing their efforts or taking the time to explore a broader range of possibilities. In our first two patterns, we examine the broader exploration side of this trade-off at two different scales. Taking the macro-scale first, we identify a pattern where users make significant progress on distinct branches of the tree (see Figure 4.3 for an example). We interpret this pattern as the user investigating multiple hypotheses about the puzzle solution, using multiple instances of the game client or *Foldit*'s save and restore features to deeply explore them all. We call this the *multiple hypotheses* pattern.

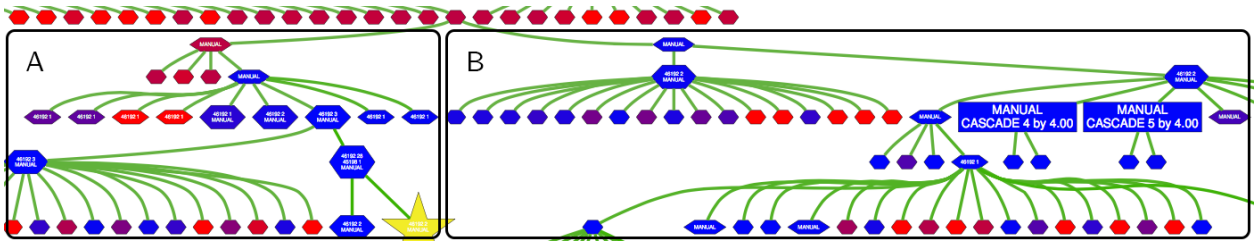


Figure 4.3: An example of the *multiple hypotheses* pattern. The two hypotheses branch out one of the nodes at the top and continue to the left (A) and right (B).

At the micro-scale, users very frequently generate a large number of possible next steps (i.e., a branch with a large number of children), but most often proceed to explore only one of them further. This is natural given the iterative refinement needed to successfully participate in *Foldit*. Hence, users that exhibit a pattern of much more frequently exploring multiple local possibilities demonstrate an unusual effort to explore more broadly. We call this the *inquisitive* pattern. Figure 4.4 shows an example of this behavior.

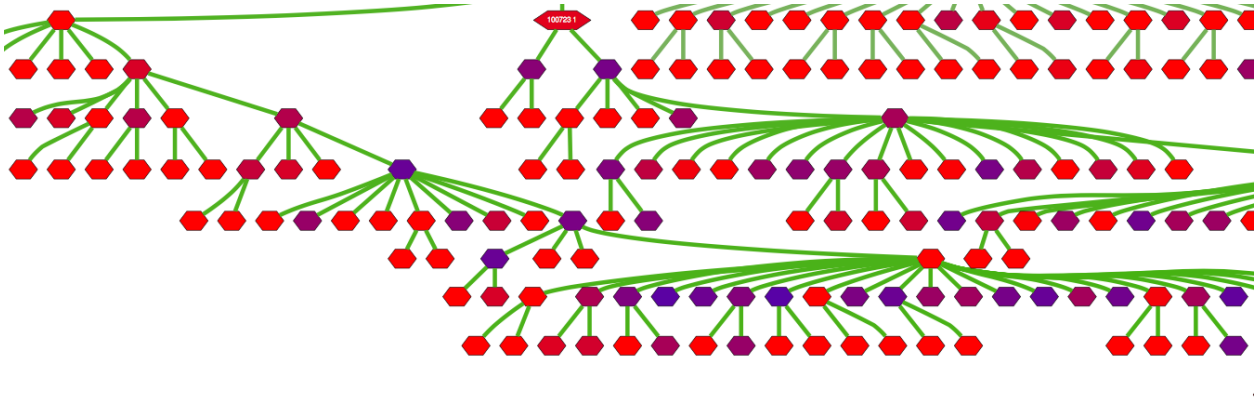


Figure 4.4: An example of the *inquisitive* pattern. Note how frequently multiple children of the same node are explored when compared to the tree in Figure 4.3.

Optimization Navigating the extremely heterogeneous solution space is the primary challenge in *Foldit*, so we look closely at how users attempt to optimize their solutions, digging more deeply into users' approaches to exploration than with the previous two patterns. We identify two related patterns describing users' fine-grained approach to optimization. The solution spaces of *Foldit* puzzles contain numerous local optima that users must escape, and we identify an *optima escape* pattern highly suggestive of a deliberate attempt to escape a local optimum. This pattern occurs when a user has a high-scoring node with a low-scoring child, and then chooses to explore from the low-scoring child. The user was willing to ignore the short-term drop in score to try to reach a more beneficial state in the long-term. Figure 4.5 gives an example of this pattern.

In the other direction, we identify the *greedy* pattern in which users exclusively explore from the best-scoring of the available options. Obviously, some amount of greedy exploration is necessary in order to refine solutions, but in its extreme form greedy exploration deserves recognition as a pattern with significant potential impact on problem-solving success. Naturally, these two patterns do not cover all the ways users explore the problem space, but they do characterize specific problem-solving structure of interest in this analysis.

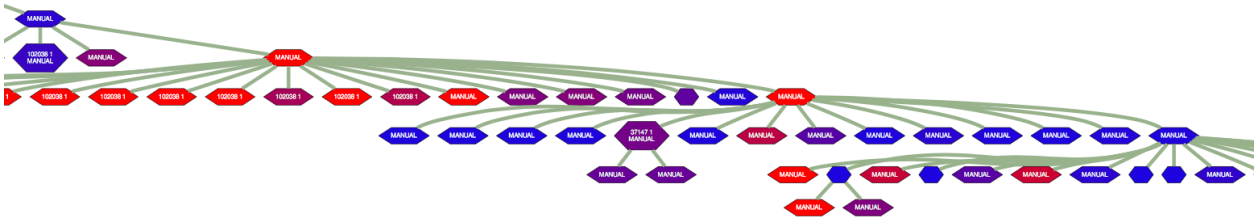


Figure 4.5: An example of the *optima escape* pattern. The user transitions from a relatively high-scoring (i.e., blue) state in the upper left to a low-scoring (i.e., red) state. What makes this an example of the pattern is that exploration from the low-scoring state. In this case, the perseverance paid off as the user reaches even higher-scoring states in the lower right.

Human-computer collaboration Human-computer collaboration is a vital part of *Foldit*, and managing the trade-off between automation and manual intervention is a key feature of solving *Foldit* puzzles. We identify two patterns that each focus on one side of this trade-off. The first, the *manual* pattern, corresponds to extended sections of exclusively manual exploration. Since recipe use is very common, extended manual exploration represents a significant investment in the manual intervention side of the trade-off. Limitations with *Foldit* logging data prevent us from capturing all the manual exploration (i.e., it is not always possible to determine whether an action was performed by a user manually or triggered as part of an automated recipe), but what can be captured is still an important dimension of variance among problem-solving behavior.

Our final pattern concerns recipe use. Some users apply a recipe to every child of a node periodically throughout their solution tree, using it as a clean-up or refinement step before continuing on (see Figure 4.6). We call this the *repeated recipe* pattern. Recipe use is very diverse and frequently doesn't display any specific structure, making this pattern interesting for its regimented way of managing some of the automation while solving.

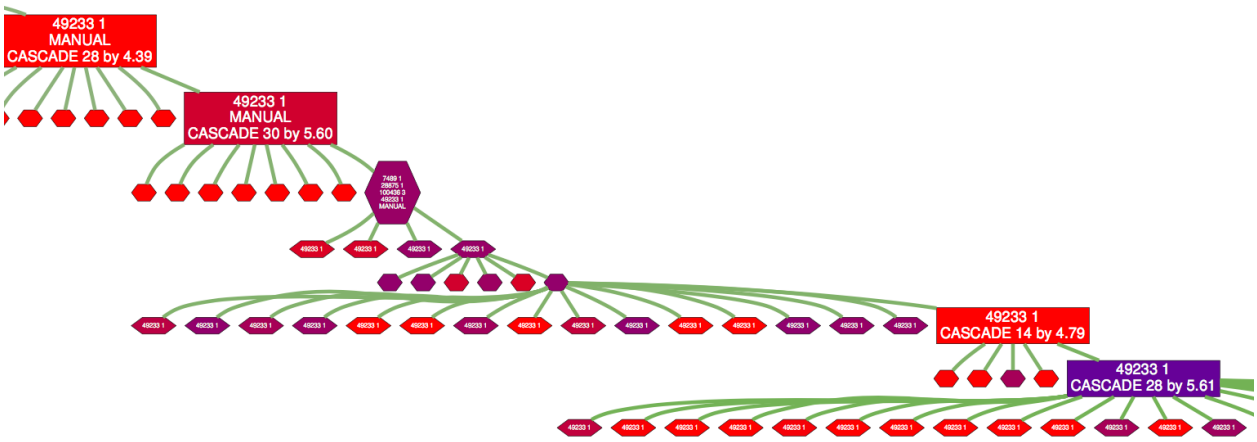


Figure 4.6: An example of the *repeated recipe* pattern. At three points in this solution tree snippet, the user applies recipe 49233 to every child of a node.

4.3.2 Patterns and User Performance

To understand how the patterns we identify relate to skillful problem-solving in an open-ended domain like *Foldit*, we compare their use among high-performing users to that among lower-performing users. Specifically, we analyze the occurrence of these patterns in the 15 best-scoring solutions from each puzzle and compare that to the occurrence in solutions from each puzzle ranked from 36th to 50th. Though it varies somewhat between puzzles, in general the solutions ranked 36th to 50th represent a *middle ground* in terms of quality. They fall outside the puzzle's state-of-the-art solutions, but remain well above the least successful efforts. Throughout these comparisons we use non-parametric Mann-Whitney U tests with $\alpha = 0.008$ confidence (Bonferroni correction for six comparisons, $\alpha = 0.05/6$), as our data is not normally distributed. For each test, we report the test statistic U, the two-tailed significance p, and the rank-biserial correlation measure of effect size r. In addition, since some of the metrics we compute may not apply to all solution trees (e.g., the tree contains no branches where the inquisitive pattern can be evaluated), we report the number of users involved in the comparison n for each test (the full sample is $n = 330$).

We find high-performing users explore more broadly than lower-performing users. For

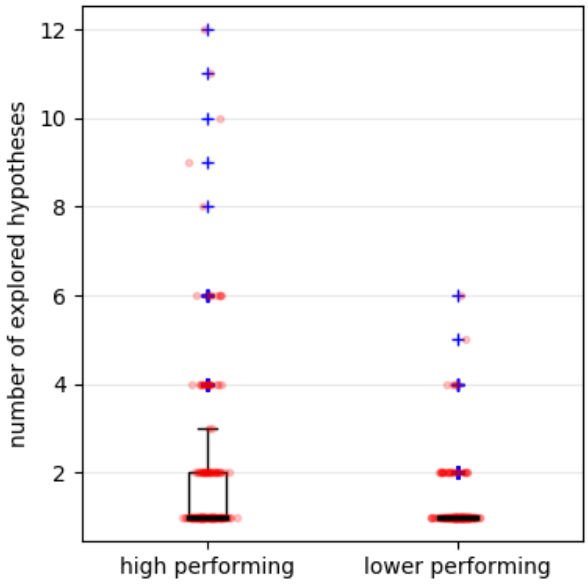


Figure 4.7: The number of hypotheses pursued in each solution tree for high- and lower-performing users. High-performing users frequently pursue two or more hypotheses, whereas lower-performing users most often pursue just one. Red circles show the distribution of individual users.

the *multiple hypotheses* pattern, high-performing users pursued significantly more hypotheses than lower-performing users ($U = 10569$, $p = 0.000014$, $r = 0.217$, $n = 330$) (see Figure 4.7). For the *inquisitive* pattern, we compute the proportion of each user's exploration that matches the pattern (i.e., of all the branches in a user's solution tree, in what fraction of them did the user explore more than one child) and find high-performing users explore inquisitively more often than lower-performing users ($U = 9343$, $p = 0.000295$, $r = 0.231$, $n = 313$) (see Figure 4.8).

We also find high-performing users work harder to avoid local optima. For the *optima escape* pattern, we compute the number of times this behavior occurs in each solution and find that high-performing users engage in this behavior more than lower-performing users ($U = 11183.5$, $p = 0.00185$, $r = 0.173$, $n = 330$) (see Figure 4.9). For the *greedy* pattern, we compute the proportion of each user's exploration that matches the pattern (i.e., of all

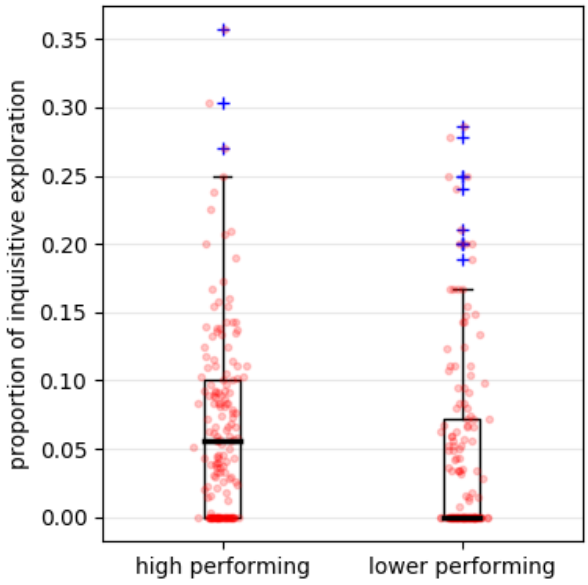


Figure 4.8: The proportion of all the branches in a user's solution tree in which the user explored more than one child for high- and lower-performing users. Red circles show the distribution of individual users.

the branches in a user's solution tree, in what fraction of them did the user only explore the best-scoring child). While high-performing users engaged in greedy optimization less often than lower-performing users, the difference was not significant ($U = 9079$, $p = 0.0158$, $r = -0.163$, $n = 295$) (see Figure 4.10).

Finally, we find no significant difference between high- and lower-performing users in the frequency they manually explore and employ recipes. For the *manual* pattern, we compute the number of manual exploration sections in each solution and find no significant difference between high- and lower-performing users ($U = 13334$, $p = 0.789$, $r = 0.014$, $n = 330$). For the *repeated recipe* pattern, we computed the median frequency of recipe use along all paths in the solution (i.e., for each path from the root to a leaf, in what fraction of the nodes did the user trigger at least one recipe) and though lower-performing users used recipes more frequently, the difference between high- and lower-performing users was not significant ($U = 11342$, $p = 0.0140$, $r = -0.157$, $n = 329$).

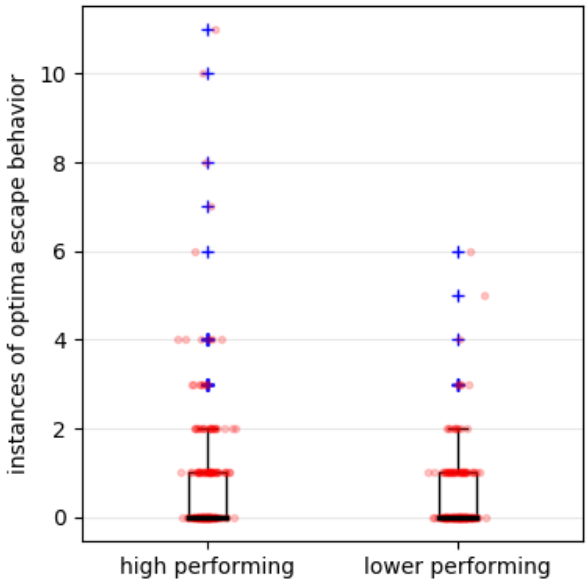


Figure 4.9: The number of times in each solution a user engages in *optima escape* behavior for high- and lower-performing users. Red circles show the distribution of individual users.

4.4 Discussion and Future Work

The results from our analysis of our solution tree visualizations illuminate some key problem-solving patterns exhibited by individual *Foldit* users. Namely, how broadly an individual explores, both on a macro- and micro-scale, how actively an individual avoids local optima by engaging in less greedy optimization and actively pursuing locally suboptimal lines of inquiry, and how an individual manages the interplay between automation and manual intervention. While our visualization methodology must contend with many domain-specific challenges, the solution tree structure is common to many problem-solving domains. Both our overall approach and the insights into users' problem-solving structure it yields have broad potential applicability.

Comparing high- and lower-performing users in their application of these patterns suggests that skillful problem-solving in an open-end domain like *Foldit* involves broader exploration and more conscious avoidance of local minima. This finding that a feature of successful

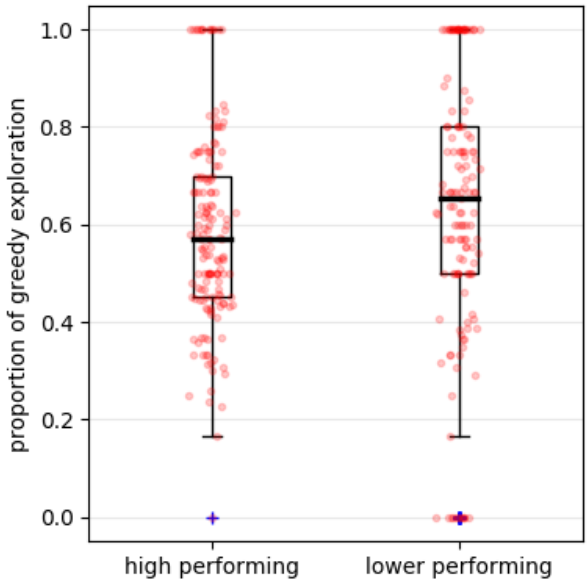


Figure 4.10: The proportion of all the branches in a user's solution tree in which the user explored only the best-scoring child for high- and lower-performing users. The fact that the median for both categories of user is above 0.5 indicates that this pattern in an important part of refining solutions in *Foldit*. Red circles show the distribution of individual users.

solving is not being enamored by the current best solution and employing effective patterns for avoiding myopic thinking has implications for the behaviors that should be taught to develop successful problem solvers. Further work is required on other large open-ended domains to confirm this trend.

The finding that users of different skill levels use greedy exploration, manual exploration, and automation in similar amounts suggests skillful deployment of non-greedy exploration, automation, and manual intervention takes place at a more fine-grained level than overall quantity. Though this work focuses on the presence or absence of specific solving patterns, the timing and sequencing of these patterns are likely to be critical to success. Further work is needed to investigate what differentiates effective and ineffective use of specific patterns.

The *Foldit* dataset itself presented significant challenges for our analysis, and we addressed these through an iterative visualization-based methodology. This process served as a design

method for generating a visual grammar to describe a complex problem-solving process. We do not study the generalization of this approach to other datasets and domains in this work, but the prerequisites for its application to other open-ended problem-solving domains can be concisely enumerated: (1) the logs of user activity establish clear temporal relationships between solution states such that those states can be visualized as a progression through the solution space, (2) the solution state or associated metadata is amenable to visual encoding, so that the visualized progressions can represent fine-grained details of the solving process, and (3) deep problem-solving domain expertise is available to provide the necessary context for interpreting and summarizing the visualized structures.

Our chosen subset of *Foldit* data represents only a small fraction of the total available data. In particular, we limited our analysis to a sample of similar prediction puzzles, and compared specific ranges of high- and lower-performing users. Though these choices are well-motivated, it is an important question for future work as to whether our results hold across different datasets and groups of comparison. More broadly, *Foldit* supports numerous variations on the prediction and design puzzle archetypes, which offers an exciting opportunity to study problem solving across a number of related contexts with varying goals, constraints, inputs, and tools.

Chapter 5

COLLABORATION

This chapter is based in part on work the author published at CSCW 2017.

5.1 Introduction

Collaborative problem solving is an integral part of many important tasks, from scientific discovery [93] to management [26]. The increasing prevalence of technologically-mediated collaboration provides an unprecedented opportunity to expand the scale, speed, and sophistication of collaboration on difficult problems. Progress in this direction could take many forms. Expanded scale and sophistication could arise from optimizing the design of collaborative mechanisms, such as how systems engage users in the problem or how problems are posed, and collaborative structures, such as how work is divided and shared. Creating layers of machine intelligence to schedule group work and dynamically adapt environmental parameters such as team size could achieve increased speed and solution quality, as has been applied to task routing in Wikipedia [13].

This opportunity is especially promising in creative, open-ended domains where good solutions are not known. It is not clear if existing models of collaboration (e.g., [3, 34]) apply to these complex spaces at scale. Hence, a better understanding of collaboration in large-scale open-ended domains is needed if intelligent problem-solving systems are to facilitate and coordinate productive collaboration. We contribute to the development of this understanding by investigating open-ended collaborative problem solving in the scientific-discovery game *Foldit*.

The structure of the collaboration in *Foldit* incorporates meaningful aspects of real-world settings. It is flexible and endogenous. Users control how and when they collaborate, where

they focus their efforts, and even whether they participate in collaboration at all. Groups are user-driven and their organization is ad-hoc rather than imposed top-down. This collaboration also occurs at multiple scales and across multiple channels. Users collaborate on individual puzzles, but may also remain an active member of a single group for years. *Foldit* facilitates direct sharing of solutions, but users also collaborate via text chat, screenshots, forums, and shared macro scripts called *recipes*.

In this work, our analysis is guided by three primary questions: (1) how do the social aspects of *Foldit* impact an individual's behavior? (2) what factors have significant impact on group success? and (3) how do users structure their collaboration in *Foldit*? The first question motivates our investigation of the effects of early collaboration and early competitive success on a user's continued participation in *Foldit* and the impact joining a group has on individual performance. We find that early collaboration and competitive success are each associated with increased participation, and that joining a group leads to increased individual performance. In service of the second question, we explore how features of group activity at difference scales correlate with group performance. We find that features measuring group collaborative skill and individual group member skill correlate strongly with group performance, that participation has moderate correlation with performance, and that group and individual experience only correlate weakly with group performance. We also examine how features of team structure predict team performance, and show that the amount of collaborative refinement matters far more than parallel exploration or a larger team size. Finally, to address the third question we produce an ontology of team structure archetypes in *Foldit* and find that more complex structures have higher average performance.

5.2 Background

Foldit users can join together in groups to tackle puzzles collectively. While users can communicate via a variety of typical online tools such as text chat and image sharing, *Foldit* provides an explicit mechanism for collaboration called *evolving*. Whenever a member of a group generates an individual solution (a *soloist* solution in *Foldit* parlance), they can choose

to share it with the other members of their group. Those other members can then import this soloist solution directly into their client, and attempt to modify and improve it. If a teammate successfully improves on the soloist effort, the new, improved solution is recorded as an *evolver* solution. An evolver solution can in turn be shared and evolved just like soloist solutions. A group's official solution for a puzzle is the highest scoring solution, soloist or evolver, produced by any member of that group. Prior work studying group behavior in *Foldit* has focused on the sharing of automated macro scripts called *recipes* [11], though there has been limited discussion of group solving dynamics [12].

There is an important distinction between a group and a team in *Foldit*. The latter is an association that enables users to collaborate. A user can choose to become a member of a group, which will enable them to work on solutions shared by other group members. Users can only be a member of one group. Groups persist across multiple puzzles, and have names and sometimes even logos. By contrast, a team is a specific set of users who work together on a specific puzzle. When a soloist shares a solution and one or more group members evolve that solution, those users form an ad-hoc team. These teams are transitory and a user may be a member of multiple teams on a single puzzle, as both a soloist and evolver.

The complexity that makes *Foldit* an attractive environment for the investigation of open-ended collaborative problem solving also presents significant challenges. First, the variety of channels over which collaboration can occur means that only a portion of the collaborative activity is directly observable in the data *Foldit* makes available. Specifically, we can observe the solutions shared via the *Foldit* client itself and through this understand who is contributing to a group effort and whether their contribution consists of a de novo effort or directly builds on work by another group member. The forums hosted by the *Foldit* project and the recipes shared among groups are also observable, though we do not analyze these channels in this work. We cannot observe the sharing of ideas or other collaboration happening over other channels such as text chat. Due to this limited picture, fine-grained analysis of collaborative acts and problem-solving strategies requires a detailed look at low-level problem-solving behavior in order to infer the larger patterns at work. This chapter looks very

broadly, focusing on the aggregate trends and correlations, with the goal of guiding future, more targeted analysis toward the most salient phenomena in need of deeper explanation.

A second challenge is the presence of many confounding variables. As *Foldit* is an active problem-solving community, randomized controlled trials and other experimentally controlled scenarios are absent from the data on its users' behavior. Hence, in assessing the impact and importance of various factors, establishing simple lines of cause and effect is frequently infeasible. We tackle this challenge in several ways. Throughout our analysis, we employ a combination of data analysis and visualization to illustrate the relevant trends and pair this with a broad discussion of the potential factors at work. This enables us to work toward an understanding of collaborative problem-solving in *Foldit* within the limitations imposed by the data. In the case of quantifying the effect of group membership on individual performance, the problem of confounding variables is particularly acute and we take the additional measure of carefully constructing a simulated controlled experiment. We identify a suitable *synthetic treatment* population (i.e., users who joined a group some time after they began playing *Foldit*), and pair each member of that population with the most similar member of the *synthetic control* population (i.e., users who never joined a group) in order to minimize the possibility that differences between the populations beyond group-joining itself are responsible for effects we observe.

5.3 Impact of Collaboration

5.3.1 Effects of early collaboration and competitive success on participation

In volunteer-based problem-solving communities such as *Foldit*'s, keeping users engaged is crucial to the success and longevity of the project. To contribute solutions to complex, open-ended problems, users must develop significant expertise, and thus must participate long enough to accomplish this. Furthermore, important social mechanisms such as collaboration and competition, both of which feature prominently in *Foldit*, require a critical mass of users to function effectively. Hence, understanding key drivers of long-term participation is

important for the design of collaborative problem-solving systems. To this end, we assess how interaction with both collaboration and competition affects participation in *Foldit*.

To address this question, we analyzed the data from all *Foldit* users across the 681 prediction puzzles released since early 2011 (puzzles released before then were not categorized). This dataset consists of 26,048 users who contributed 179,723 solutions. Since a user's initial experiences in *Foldit* must necessarily play a significant role in their choice of whether to continue participating, we group the population in our dataset according to the presence of two experiences in each user's first five puzzles.

Specifically, we consider a user to have experienced *early success* if in any of their first five puzzles, their soloist solution ranked in the top 25 soloist solutions. We select the top 25 solutions as a threshold because that is how many users are shown on the first page of ranked soloist solutions for each puzzle on the *Foldit* website, and ranking is the primary mechanism for social recognition in *Foldit*. In other words, we consider a user to have experienced early success if they can see themselves on the first page of ranked soloist solutions for any of their first five puzzles. We consider a user to have experienced *early collaboration* if in any of their first five puzzles, they participated as a member of a group. *Foldit* has 3–7 puzzles available for users to contribute to at a time (individual puzzles expire and get replaced on a timeline of 1–2 weeks), so a user's first five puzzles reasonably approximates the content available at the time they begin participating. These criteria give us the four non-overlapping classes listed in Table 5.1.

In terms of quantifying participation, the most relevant measure is also the most straightforward: the number of puzzles a user contributed to. Therefore, we compare the number of puzzles contributed to by users in each of the four classes.

As Figure 5.1 shows, both early collaboration and early success are associated with increased participation. Each curve visualizes the rate at which users in a given class stop contributing to *Foldit*. For example, almost 40% of those who had both early success and early collaboration contributed to at least 100 puzzles, while only 29% of those with only early success did so. We use a Mann-Whitney U test for non-normally-distributed data to test

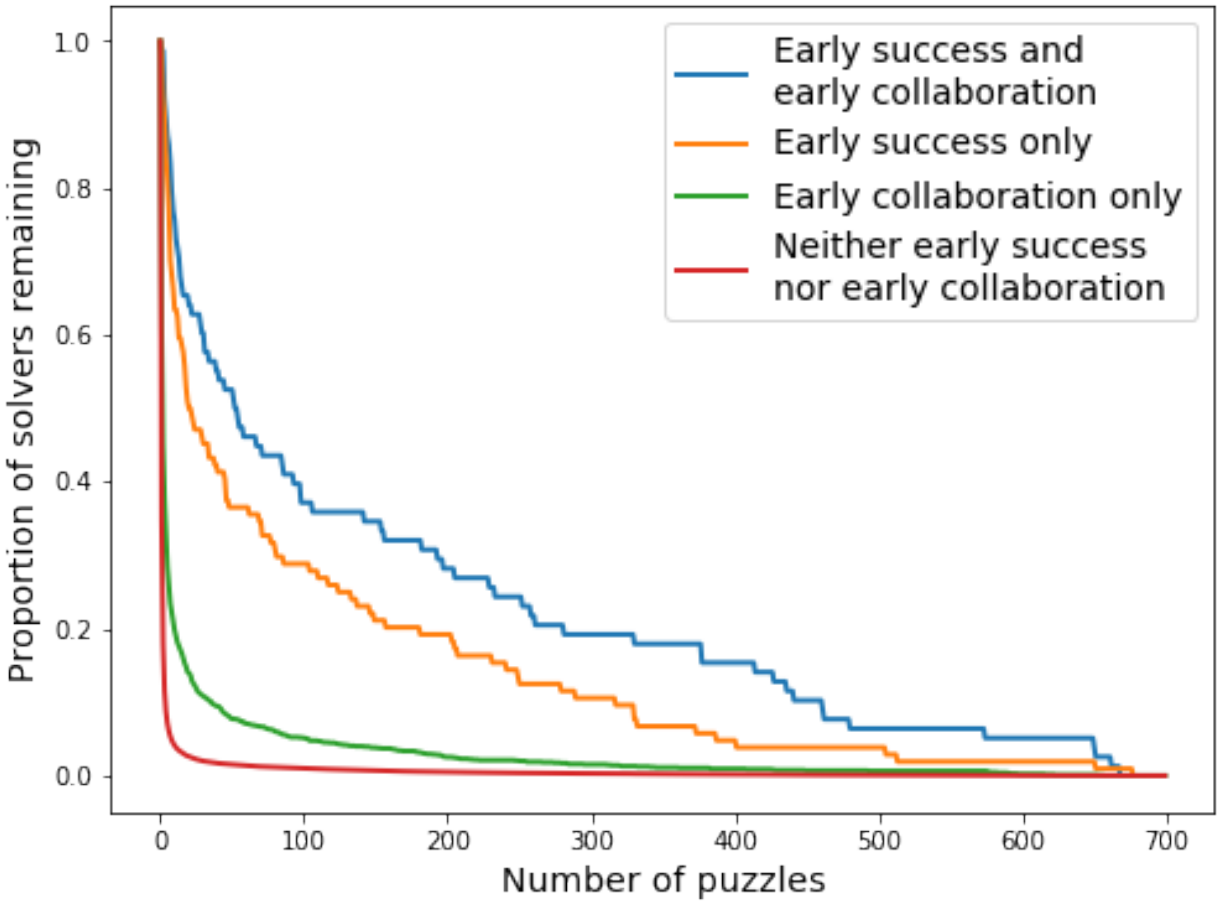


Figure 5.1: The effects of early collaboration and success on continued participation. For a given number of puzzles on the x-axis, each curve indicates the proportion of users in that class who contributed to at least that many puzzles. This figure shows an association between increased participation and early success and early collaboration, with early success having the greater association. It is notable that early collaboration is associated with increased participation for users with and without early success.

significance and rank-biserial correlation coefficient (r) to measure effect size of the differences between classes (we use multiple Mann-Whitney U tests instead of a Kruskal-Wallis test in order to understand the magnitude of the effects). Specifically, we compare each class to the next best class in terms of participation (i.e., the class with early success and early collaboration is compared to the class with early success only; the class with early success

only is compared to the class with early collaboration only, and so on). Summary statistics for each class and the results of our statistical comparisons are given in Table 5.1.

	users	mean puzzles	median puzzles	effect size
Success and collaboration	78	149.5	53	0.188*
Success only	104	94.8	21	0.649***
Collaboration only	1697	20.4	2	0.336***
Nosuccess or collaboration	24169	5.1	1	—

Table 5.1: The summary statistics for each class and results of statistical comparisons between classes. The rank-biserial correlation measure of effect size is given for the comparison of each row with the class on the next row (hence why the last row omits these). The p values for Mann-Whitney U tests of the difference in participation between rows are indicated by: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

While early success was associated with greater participation than early collaboration, it is notable that early collaboration is associated with significant increases in participation for users independent of whether they experience early success. These results are consistent with existing research on participation in online communities, in particular the findings that recognition of user contributions (e.g., early success) and emphasis of social context (e.g., early collaboration) can increase participation [52].

It is also worth noting that though users with early success contribute to many more puzzles on average, they still stop contributing at an appreciable rate. In some sense, *Foldit* is failing to capture the talent of *all* its promising new users. It may be possible that well-designed and well-timed feedback could convert more of these users to long-term contributors. From this perspective, early success and collaboration could serve as indicators for identifying users with greater potential to become long-term contributors, and help guide attention from facilitators or the system itself toward integrating them into the community.

Though we identify significant differences in participation between groups that experience early success and collaboration and those that do not, we cannot establish clearly delineated cause and effect. It is possible that those who early on chose to join a group or put in the work to get a high score are already disposed to greater participation. Regardless of the dynamics

at work, collaboration and competition can clearly provide experiences or opportunities with potentially long-term effects on the participation of new members. Designers of collaborative problem-solving systems interested in increasing participation should explore ways of both increasing the prevalence of these experiences and integrating them more tightly into the fabric of their systems.

5.3.2 Effect of group membership on individual performance

The literature on learning and collaboration makes clear that collaboration can have significant learning benefits in both traditional and computer-mediated settings [68, 85]. It is not clear, however, if this effect extends to open-ended problem-solving domains such as *Foldit*. In *Foldit*, as in many other settings, developing user skill is essential, and quantifying the role of collaboration in this process is necessary to construct a comprehensive understanding of the problem-solving ecosystem. Furthermore, characterizing the impact of group membership on individual performance in detail could serve as a guide to future analysis of the specific mechanisms at work.

Given the number of confounding factors surrounding group membership in *Foldit*, isolating its effect on individual performance requires a carefully designed comparison. Ideally, we would have two randomly assigned, otherwise identical subsets of *Foldit* users where the members of one subset joined a group and members of the other subset did not. Since our data comes from an active scientific-discovery game rather than a controlled lab setting, such an ideal scenario does not exist. Hence, we construct two subsets of users from the available data in such a way as to control confounding factors as much as possible.

In particular, we construct a *synthetic control* sample that never joins a group and a *synthetic treatment* sample that does. To measure the effect of group membership on individual performance, we compare how the performance of members of each sample develops before and after those in the synthetic treatment sample joined a group. We construct our two samples as follows. First, we identify the subset of users for the synthetic treatment sample who will support a robust comparison of their performance before and after they first

join a group. Specifically, we select users who began not part of any group, who contributed to more than 30 puzzles and at least 10% of their total puzzles before they joined a group, and who contributed to at least as many puzzles after first joining a group as they did before joining any group. These criteria ensure a sufficient demonstration of each user's performance before and after they joined a group. There are 92 users that meet our criteria.

In order to ensure the validity of comparisons between the synthetic treatment and control samples, we control for two potentially confounding variables in constructing the synthetic control sample. In particular, we construct the synthetic control sample by pairing each user in the synthetic treatment sample with a user that never joined a group minimizing the differences between each pair of users along two dimensions. First, we minimize the difference within each pair of performance before the synthetic treatment sample user joined a group, measuring performance as the ratio of the user's solution score to the score of the best soloist solution. Minimizing performance differences before treatment occurs helps ensure that any differences that emerge after the treatment can be attributed to the treatment itself. Second, we minimize the difference within each pair in the total number of puzzles each user contributed to. Here we use overall participation as a proxy to control for differences in overall engagement. As Figure 5.2 shows, this process results in very similar distributions of pre-treatment performance and overall participation for our synthetic treatment and control samples. Users in our synthetic treatment sample are diverse in terms of when they joined a group, as shown in Figure 5.3

To measure the effect of group membership on individual performance, we compare both the number of users who experienced an improvement in performance and the magnitude of the improvement in each of our two samples after treatment occurs. To compute the magnitude of improvement, for each pair of users, we measure their median performance over the same number of puzzles following the treatment as the number they contributed to prior to the treatment. For example, if a user in the synthetic treatment sample contributed to 40 puzzles before joining a group, we measure their median performance and the median performance of the synthetic control sample user they were paired with over their first 40

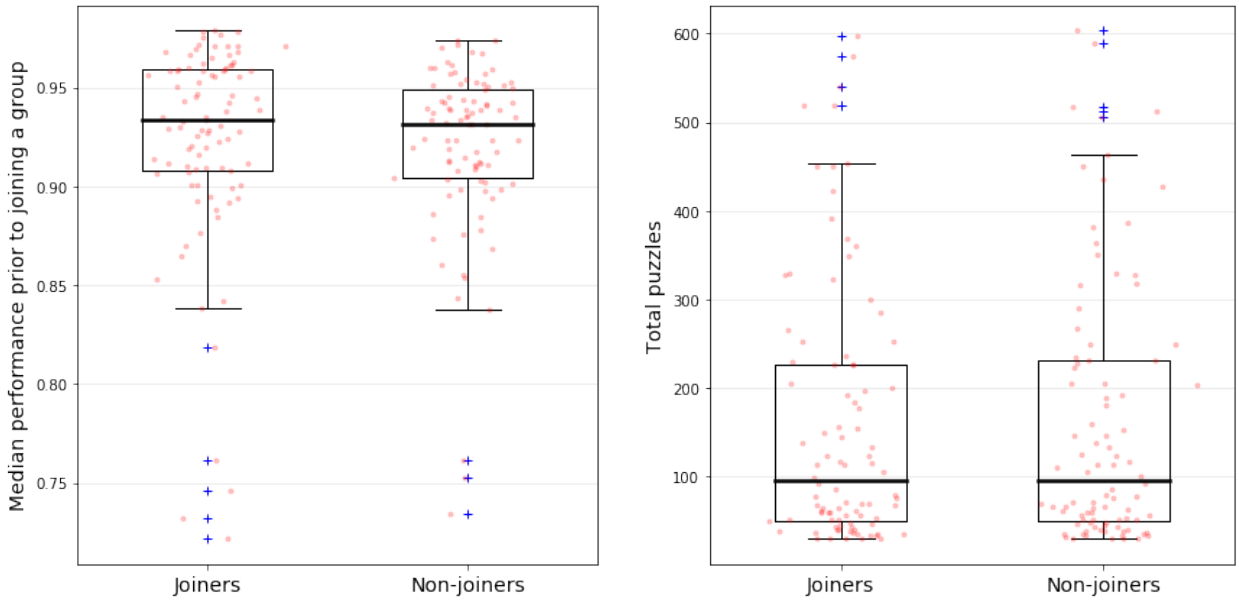


Figure 5.2: The distribution of median performance and overall participation for our synthetic treatment (i.e., group-joining) and control (i.e., non-group-joining) samples before treatment users joined a group. Red circles indicate individual users.

puzzles and then over their next 40 puzzles (i.e., their 41st through 80th puzzles). Then, we compute the difference between each users' median performance after the point of treatment and their median performance before treatment.

Comparing the number of users who experienced an improvement in performance in each sample (i.e., the user had higher median performance after the point of treatment than before), more users improved in the synthetic treatment condition (58) than in the synthetic control condition (40). Pearson's χ^2 test indicates this can be attributed to a significant difference between the two conditions ($\chi^2(1, N = 184) = 6.31, p = 0.012$). Using a Mann-Whitney U test for non-normally-distributed data to test significance and rank-biserial correlation (r) to measure effect size, we find that among users who experienced improvement, those in the synthetic treatment sample improve their performance more than those in the control sample, but that this difference is not statistically significant. The 58 synthetic treatment users who improved have a mean increase in performance of 0.036 (median of 0.025) compared to a

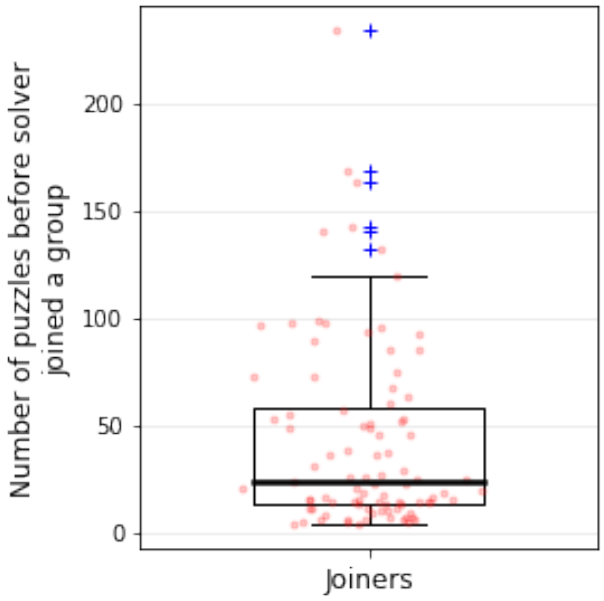


Figure 5.3: The distribution of how many puzzles users in our synthetic treatment sample contributed to before joining a group. Red circles indicate individual users.

mean increase in performance of 0.023 (median of 0.018) for the 40 synthetic control users who improved ($U = 906$, $p = 0.067$, $r = 0.219$).

The finding that collaboration improves individual performance in *Foldit* is not surprising, but serves as useful confirmation that this effect extends to ad-hoc collaboration on complex, open-ended problems in an entirely virtual environment. In particular, improvement was more widespread among group members than among non-group members. This finding motivates questions for deeper analysis into what about group membership in *Foldit* is contributing to the observed benefits. Any number of dynamics could be at work, including mentorship, exposure to new techniques and ideas, access to recipes shared with group members, or increased effort due to social motivation. Investigating these dynamics is important for developing a comprehensive understanding of the collaborative problem-solving ecosystem, and could provide useful insights for designers of collaborative problem-solving systems.

An interesting aspect of these results is that many users in both samples had decreased

performance after the point of treatment. This is not surprising as our metric for performance is relative rather than absolute, meaning if the user community improves as a whole over time, overall performance should appear roughly the same. Nevertheless, this lack of improvement raises the question about what intervention apart from group membership might give these users the kind of boost experienced by the majority of those that joined a group.

Though we have taken steps to minimize the effect of confounding variables on our results, it is not possible to guarantee their absence. Regardless of any precautions, we are still necessarily comparing users who chose to join a group to those who did not, making our results vulnerable to selection bias. Due to our method of constructing our samples, however, any confounding factors that do play a role are not evident in user performance pre-treatment or in overall participation. Furthermore, we measure improvement over the same number of puzzles for both conditions, so we control for increase in engagement associated with joining a group in terms of its effect on participation. Hence, we have confidence that group membership plays a significant role in the observed effect.

5.4 Group Performance

As in so much of real-world problem solving, group performance is a key driver of high quality solutions in *Foldit*. Developing a better understanding of what contributes to group success and failure could play a vital role in improving outcomes. This understanding is a necessary foundation for designing systems that structure group work in ways that enable more effective collaboration, or implementing layers of machine intelligence to schedule individual efforts in the most effective combination. Hence, we explore the relationship between group performance and a variety of factors. While the relationships we explore are correlational rather than explanatory, they provide a lay of the land, and, more importantly, generate questions that can guide future investigations. A deep understanding of group performance in *Foldit* will require fine-grained analysis of collaborative acts and the specific collaborative problem-solving behavior involved in high-quality solutions.

To conduct this exploration, we use data on group contributions on all 681 prediction

puzzles released since early 2011. Of the 451 groups that participated in at least one of these puzzles, we restrict our analysis to the 66 groups that participated in at least five puzzles with at least two group members participating in each puzzle. These 66 groups contributed 13,471 solutions (the members of these groups contributed far more solutions than this, but *Foldit* considers a group's solution to be the best soloist or evolver solution contributed by one of its members).

With this dataset, we explore group performance at both macro- and meso-scale. Specifically, we explore how features of a group's overall tenure correlate with overall group performance and how features of a group's effort on an individual puzzle correlate with performance on that puzzle. In both cases we measure performance as the ratio of a group's solution score to the score of the best group solution. As part of exploring overall performance, we also examine how these features differ between *high-performing* groups and other groups in our dataset. Given that a nuanced identification of high-performing groups would rely on the kind of deep understanding of group performance motivating our current exploration, we use a very simple threshold: we designate a group as *high-performing* if it contributed the single best solution (i.e., was ranked first) on at least one puzzle. Under this threshold, 15 of the 66 groups are considered high-performing. We view this threshold as providing a good upper bound on the number of high-performing groups — it is very unlikely that any group making a substantive collaborative contribution through *Foldit* has *never* contributed a puzzle's highest-scoring solution. In the context of an initial exploration, we view a broad, inclusive threshold as preferable to a narrow, overly-restrictive one.

Macro-scale features We explore how the following features of a group's entire solving history relate to overall group performance. We apply these to the 66 groups in our dataset.

Group experience: the total number of puzzles the group has contributed to. Groups may improve their collaboration over time, which in turn may increase their performance.

Collaborative skill: the proportion of puzzles for which the group's solution was contributed by an evolver. Collaboration in *Foldit* consists of an ad-hoc back-and-forth as

group members view, borrow from, and improve upon (i.e., evolve) each others' solutions, so no single feature will completely capture this complex process. As this feature measures the frequency with which a group's solution definitively represents an effort beyond what any individual member contributed alone, we hypothesize it approximates part of a group's overall ability to collaborate productively.

Group participation: the median number of group members participating on each puzzle over the group's entire history. Without sufficient participation, groups may be unable to benefit from collaboration.

Meso-scale features We explore how the following features of a group's effort on an individual puzzle relate to group performance on that puzzle. We apply these to the 13,471 group contributions in our dataset.

Individual experience: the number of puzzles contributed to by the most experienced participating group member (only counting puzzles prior to the one in question). Since a group's solution is the best among those produced by all its members, we similarly use the most experienced participating group member as a measure of the experience the group brought to bear on a particular puzzle.

Individual skill: the median soloist performance of the best-performing participating group member (across all puzzles that group member has contributed to up until this point). As with the previous feature, the nature of group solutions in *Foldit* motivates our considering only the best-performing participating group member.

Soloist participation: the number of group members contributing as soloists. More soloists may provide a group with greater diversity of ideas and increased ability to pursue multiple approaches to a puzzle.

Evolver participation: the number of group members contributing as evolvers. More evolvers may enable a group to better or more quickly refine its solutions.

We first describe our exploration of our three macro-scale features. For each feature, we

plot the value of that feature for each group versus that group's median performance across all puzzles it contributed to, using color to distinguish top groups from other groups. Group experience and group participation are shown in Figure 5.4 and collaborative skill is shown in Figure 5.5. In addition, we compute Spearman's rank correlation coefficient (Spearman's ρ) for each feature to measure its correlation with group performance, doing so separately for top groups and other groups. Finally, we perform for each feature a Mann-Whitney U test to determine if the difference between top groups and other groups is statistically significant, and compute the rank-biserial correlation coefficient r to measure the effect size for this test. We use these statistical measures as they are non-parametric and thus appropriate for non-normally-distributed data. These values are given in Table 5.2.

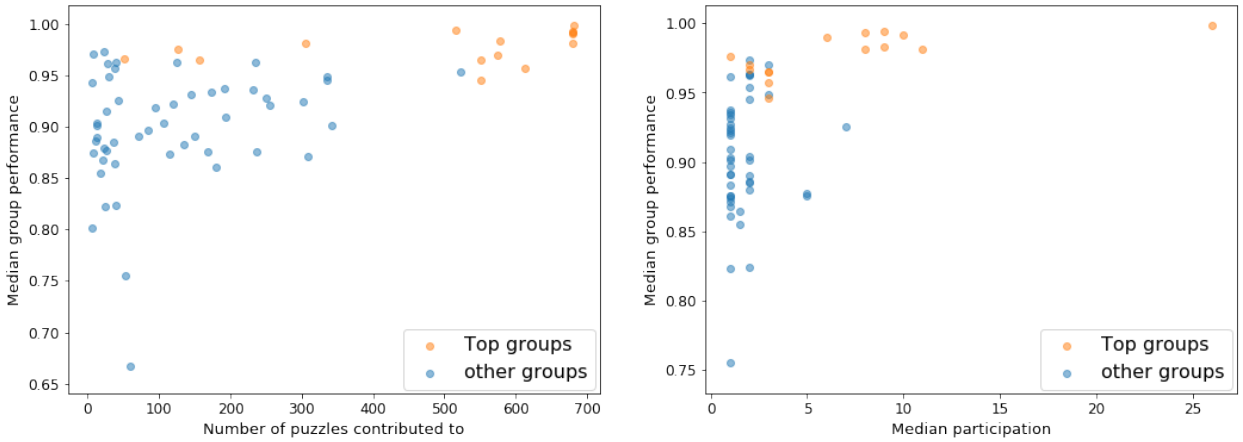


Figure 5.4: The group experience (left) and group participation (right) features versus median group performance.

We find a significant difference between top groups and other groups for all three features. We also find significant correlation for top groups between those same three features and group performance, although at differing levels of significance. The collaborative skill feature has the largest effect size and the strongest correlations, including, unique among our three features, a significant correlation with group performance among non-top groups.

As with our macro-scale features, for each of our meso-scale features we plot the value of

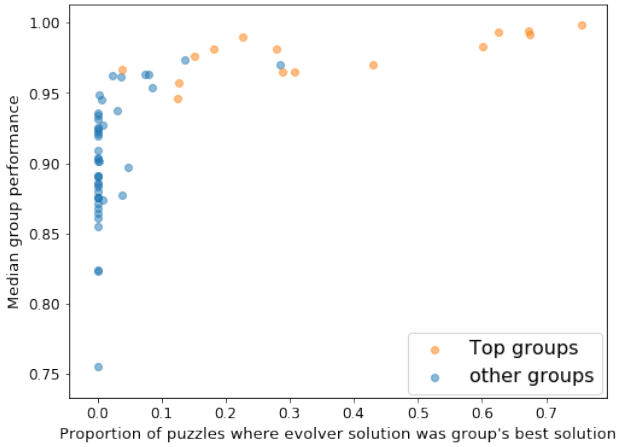


Figure 5.5: The collaborative skill feature versus median group performance.

	Top groups		Other groups			
	median	ρ	median	ρ	U	effect size r
Group experience	574	0.517*	71	0.212	69	0.820***
Collaborative skill	0.288	0.743***	0.000	0.586***	14	0.963***
Group participation	6	0.706**	1	0.140	81.5	0.787***

Table 5.2: The statistical results of our exploration of macro-scale features. For each feature, we list the median value and Spearman's rank correlation coefficient (ρ) for both top groups and other groups. We also list the test statistic U and rank-biserial correlation measure of effect size r for a Mann-Whitney U test of the difference between top groups and other groups. The p values for the correlation and Mann-Whitney test are indicated by: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

that feature for each group contribution (i.e., a separate data point for each puzzle each group participated in) versus the performance of that group on that puzzle. Individual experience and individual skill are shown in Figure 5.6 and soloist participation and evolver participation are shown in Figure 5.7. In addition, we compute Spearman's ρ for each feature to measure its correlation with group performance. These correlations are given in Table 5.3.

We find significant correlation for all four meso-scale features, though the correlation is much weaker for individual experience than the other features. As with the macro-scale features, the measure of skill has the strongest correlation with performance. We note that

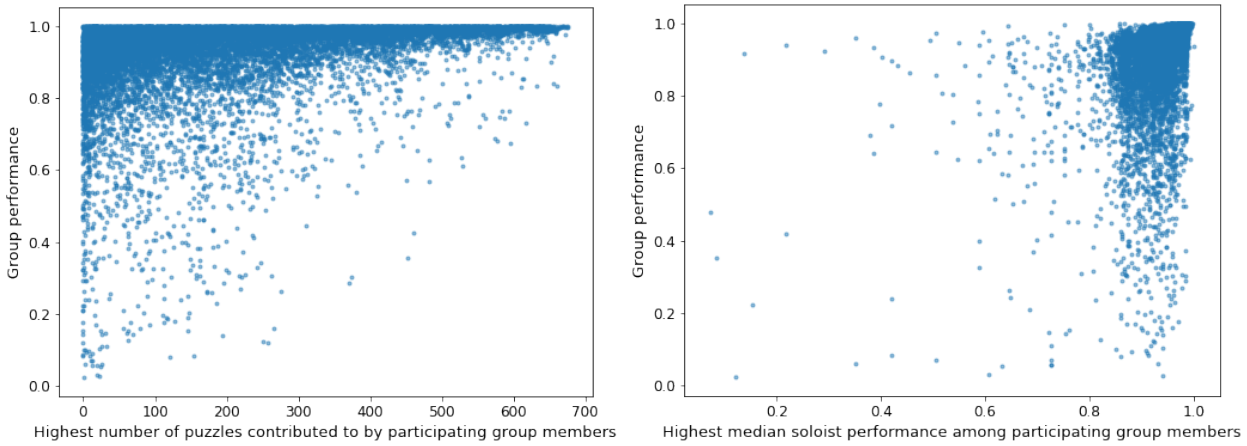


Figure 5.6: The individual experience (left) and individual skill (right) features versus group performance.

	ρ
Individual experience	0.405***
Individual skill	0.725***
Soloist participation	0.687***
Evolver participation	0.648***

Table 5.3: The statistical results of our exploration of meso-scale features. For each feature, we list the Spearman's rank correlation coefficient (ρ). The p values for the correlations are indicated by: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

participation for both soloists and evolvers appears to have diminishing returns. Using the median performance at each level of participation as a guide (shown as red hashes in Figures 5.7), we observe the benefits of increased participation diminish at more than six soloists and more than five evolvers.

The relationship we find between collaborative skill and group performance is consistent with other work on solution quality in collaborative environments. Studies of collaboration in settings including *MathOverflow* [81] and *League of Legends* [42] find that collaborative acts improve group performance. In the context of *Foldit*, this finding highlights a promising focus for future, fine-grained analysis: investigating the collaborative organization and strategies

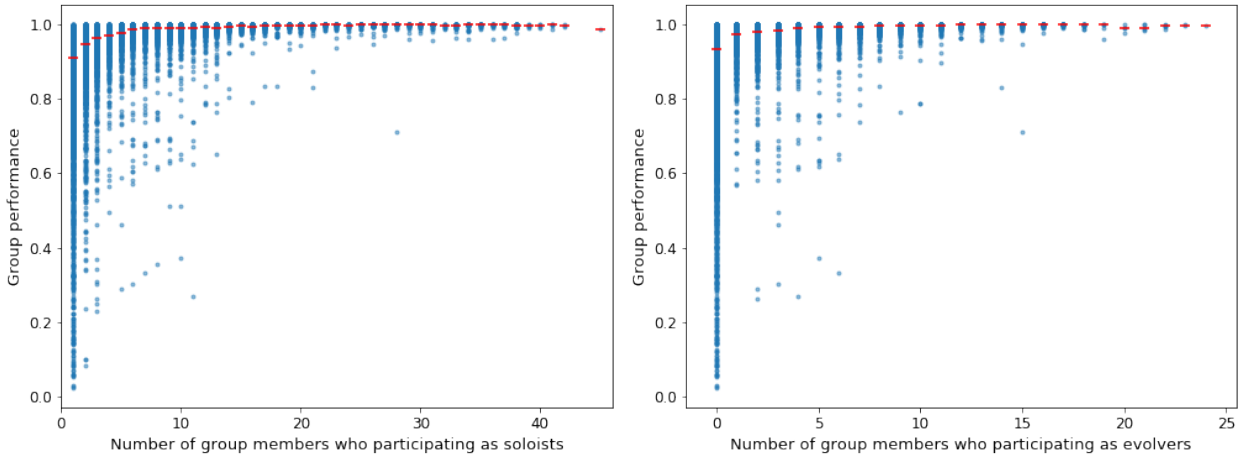


Figure 5.7: The soloist participation (left) and evolver participation (right) features versus group performance. Red lines indicate the median performance for each level of participation.

used by groups when their best solution is an evolver solution. Understanding these events at the level of user actions, including how solutions are shared and refined within a group, will be critical for characterizing why they are associated with long-term group success.

The relatively weak correlation between experience and group performance is surprising, especially since experience is often the strongest predictor of performance in a game [42]. One possible explanation is that the changing *Foldit* environment in terms of new puzzles and tools being added lessen the typical benefits of experience. Additional labeling of the puzzles in the *Foldit* dataset to better account for these differences would be one way to enable an investigation of this potential explanation. Another, more troubling explanation is that *Foldit* users are not learning from experience the way we would expect. Given the known importance of immediate feedback and reflection for learning [14], the lack of these practices as integral parts of *Foldit* may be contributing to reduced user learning. Fortunately, existing work on improving crowdsourced solutions offer models of how reflection (in the form of self-assessment [17]) and real-time feedback (in the form of high-level expert guidance [6]) can be incorporated into online problem-solving environments. Our exploration suggests there may be an opportunity for these techniques to improve solution quality in the context

of scientific-discovery games.

Participation's correlation with group performance in *Foldit* is exciting because participation in online communities is well-studied in the literature [52]. We observe that median participation is quite low for many groups, and the puzzle-level correlation of participation with group performance suggests increased participation could improve solution quality for those groups. A variety of mechanisms to increase participation have been identified, including sending personalized introductory messages emphasizing social interaction and explaining to members the value of their contributions [52]. The diminishing returns to increased participation we observe are consistent with analysis of collaboration among editors of Wikipedia articles [44]. Kittur and Kraut found that appropriate coordination techniques were necessary in order to benefit from adding more editors. A deeper look into the collaboration of *Foldit*'s top groups is necessary to reveal the coordination techniques they employ to capitalize on the benefits of higher participation.

Our initial exploration of group performance in *Foldit* is not without limitations. The thresholds we use to determine inclusion in the dataset and to differentiate top groups from other groups are clearly imperfect. Despite the requirement that every group in our dataset participate in at least five puzzles with at least two members, many of the groups appear to mostly participate with a single member (see the large number of groups with a median participation of one in Figure 5.4 and zero best solutions from evolvers in Figure 5.5). As for top groups, there are a handful we label as top groups that, at least by the features we measure, look a lot more like non-top groups. One result of our exploration is highlighting this diversity of group composition and behavior in *Foldit*.

5.5 Ontology of Team Structures in Foldit

So far in this chapter we've focused on *Foldit* groups, the large, persistent associations that allow users to evolve solutions shared by other group members. All collaboration on specific solutions, however, occurs when users improve upon others' work as members of a team. A *Foldit* team forms when a member of a group shares a soloist solution and other group

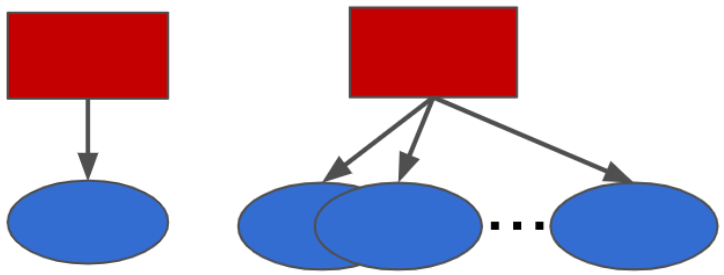
members successfully evolve it. In the remainder of this chapter, we turn our focus to the structure of these ad-hoc teams.

In the management literature, *team structure* is often defined as intra-team relationships affecting task allocation and the authority and responsibilities of team members (e.g., [77]). In the context of *Foldit*, we do not use team structure to refer to factors such as *interdependence* or *team self-leadership* management scholars use to characterize these relationships. Instead, we examine team structure in terms of the explicit organization of soloists and evolvers within ad-hoc teams. These structures emerge organically as users choose which solutions to share and which solutions to evolve over the course of a puzzle.

In this section we address a fundamental question about collaboration in *Foldit*: what kinds of team structures occur and how do they perform? Answering this question has relevance beyond contributing a better understanding of collaborative problem solving in *Foldit*. The nature of *Foldit* teams where solutions are handed off from one user to another, beginning with a single individual effort, map well onto any problem where team members are collaborating to produce a design or perform an analysis. Many online settings feature emergent teams and users autonomously engaging with work on a changing solution. The soloist-evolver model also has resonance with planning or command and control settings, with the soloist playing the role of a commander or initiator and evolvers corresponding to subordinates or implementers.

We use a dataset of 209 prediction puzzles released between 2015 and 2017 to investigate team structures occurring in *Foldit*. Across these puzzles, 880 users formed 2,454 teams. From the mechanics of collaboration in *Foldit*, we know that the minimal possible team consists of a soloist and a single evolver. We also know that multiple evolvers can work off of a single shared solution. Based on this, we identify two basic archetypes of *Foldit* team structure: the *minimal* archetype and the *one-branch* archetype. These are diagrammed in Figure 5.8, with arrows representing when a user evolves another user's shared solution.

To build a full ontology of team structures in *Foldit* beyond the two basic archetypes, we visualize the structures of all teams not corresponding to the minimal or one-branch archetypes.

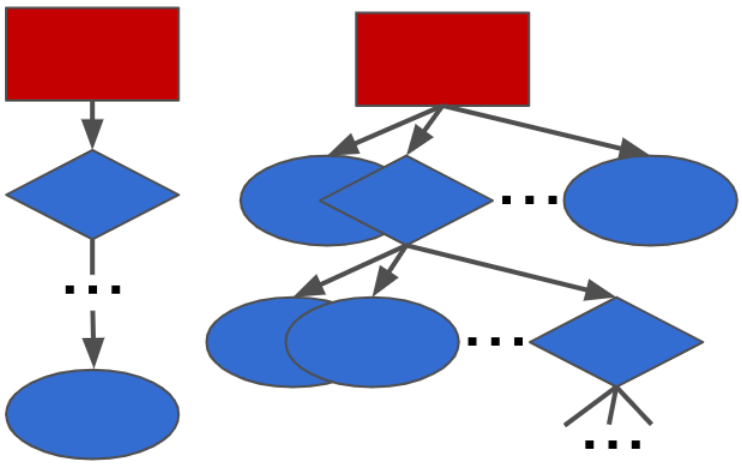


(a) Minimal archetype (b) One-branch archetype

Figure 5.8: Two basic archetypes of *Foldit* team structure. Each node corresponds to a team member, with arrows indicating when a solution shared by one member is evolved by another. Red rectangles represent soloists and blue ovals represent evolvers.

Through a careful survey of these visualizations, we identify three other archetypes that encompass all remaining team structures occurring in *Foldit*. First, there is the *line* archetype, an extension of the minimal archetype where the evolver shares their solution and another team member improves upon it. This chain of evolvers could extend for many such handoffs, as indicated by the ellipses in Figure 5.9a. Second, extending the one-branch archetype results in the *n-branch* archetype, as shown in Figure 5.9b. One of the evolvers from the first branch shares their solution, which is worked on by multiple evolvers, one of which may again share their work, and so on. The one-branch and n-branch archetypes feature parallel work from a single starting point, with evolvers working off the same shared solution in each branch. The final archetype, the *rich* archetype, captures the remaining team structures that include parallel work from different starting points. As Figure 5.10 shows, it is possible for multiple evolvers to share solutions and for the team's work to proceed on multiple fronts. Once a team reaches this degree of structural complexity, we do not see any benefit from additional fine distinctions, so we include all such structures under a single archetype.

We also identify a key variation for some team structures. In all five previous archetypes, the soloist contributes an initial solution, but is otherwise uninvolved in the collaborative process. It is possible, however, for that user to continue participating in the collaboration

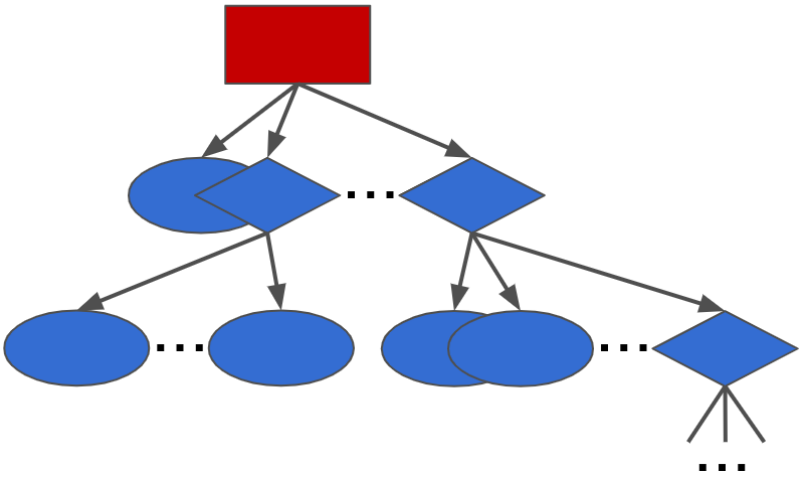


(a) Line archetype (b) N branch archetype

Figure 5.9: Two archetypes of *Foldit* team structure extending the minimal and one-branch archetypes, respectively. Each node corresponds to a team member, with arrows indicating when a solution shared by one member is evolved by another. Red rectangles represent soloists, blue ovals represent evolvers, and blue diamonds represent evolvers who shared their solution.

as a soloist. This takes the form of additional shared solutions, contributing the results of further individual work to the team effort. When these additional solutions are also evolved, the collaboration takes on a conversational, back-and-forth structure as evolvers improve upon successive versions of the soloist's solution. We call this the *iterative* variation of an archetype. Our dataset contains iterative versions of the one-branch and n-branch archetypes (see Figure 5.11) as well as the rich archetype (see Figure 5.12).

We examine the relative occurrence of the eight archetypes in our team structure ontology in Figure 5.13. For each archetype, it shows the distribution of what proportion of teams fit that archetype on each of the 209 puzzles in our dataset. For example, the fraction of minimal archetype teams varies between 0 and 50 percent, with most puzzles having 10 to 40 percent minimal teams. We note the overall stability of the populations of different team structures. Almost no puzzles have more than 50 percent of any one structure archetype. A minimal archetype team is present on 91 percent of puzzles, 94 percent of puzzles have a

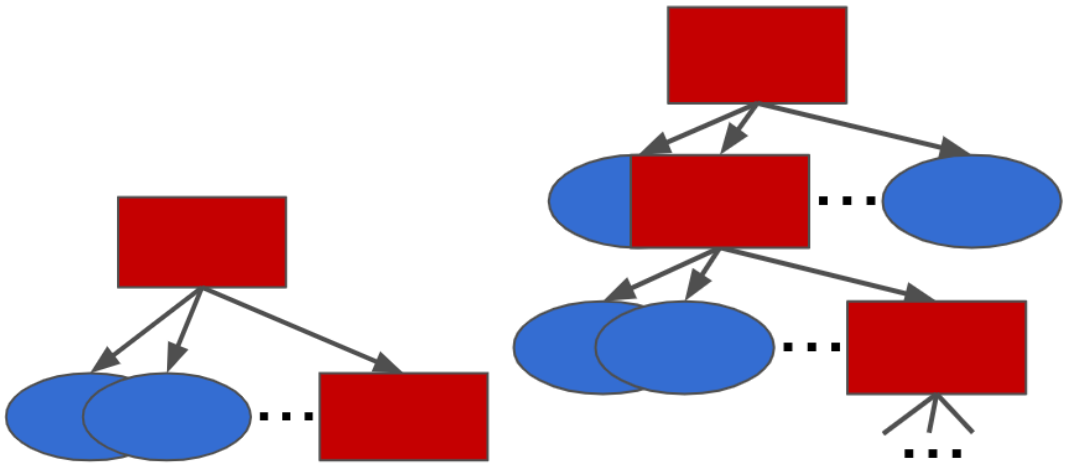


Rich archetype

Figure 5.10: The most complex archetype of *Foldit* team structure. Each node corresponds to a team member, with arrows indicating when a solution shared by one member is evolved by another. The red rectangle represents the soloist, blue ovals represent evolvers, and blue diamonds represent evolvers who shared their solution.

one-branch archetype team and 93 percent of puzzles have an iterative n-branch archetype team. The median puzzle has 20–25 percent each of minimal, one-branch, and iterative n-branch teams and 10 percent each of one-branch iterative, n-branch, and iterative rich teams. The stability of team structure populations across this large dataset suggests that on hard, open-ended problems, there is no dominant team structure. Instead, trade-offs, available resources, and specific problem context likely all influence which structures tend to emerge. This is consistent with Structural Contingency Theory [16], which states that no single team structure is optimal in all circumstances.

In addition to relative occurrence, we examine the trends in how teams of different archetypes perform. The performance of a team is the quality of the best solution contributed by any team member as part of the collaboration. Figure 5.14a shows the distribution of performance for each archetype. One notable property of team performance in *Foldit* is the very high floor. Even teams in the lowest-performing archetype, minimal, have a median performance of 0.924. Given the mechanics of collaboration in *Foldit*, this is not surprising.



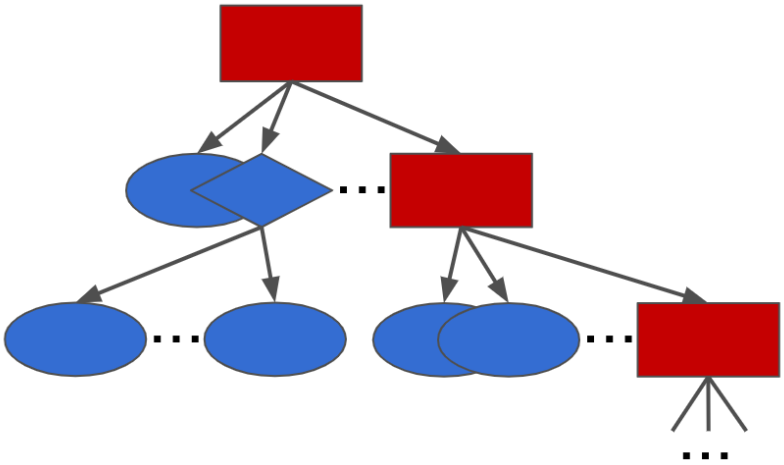
(a) Iterative one-branch archetype (b) Iterative n-branch archetype

Figure 5.11: Iterative versions of the one-branch and n-branch archetypes of *Foldit* team structure. Each node corresponds to a team member, with arrows indicating when a solution shared by one member is evolved by another. Red rectangles represent soloists, blue ovals represent evolvers, and blue diamonds represent evolvers who shared their solution. Note how the soloists share multiple solutions, continuing their participation in the collaboration past their initial shared solution.

Every collaboration must begin with a soloist choosing to share their solution. If users are more likely to share solutions with good scores, this would introduce a bias toward beginning collaboration with a relatively strong solution.

The overall trend in performance indicates that more complex team structures perform better. To establish this trend with more precision, we perform the six comparisons using a Mann-Whitney U test with an α of 0.00008 (an initial α of 0.001 with a Bonferroni correction for 12 comparisons) listed in Table 5.4. While the iterative versions of one-branch and n-branch do not significantly outperform the non-iterative archetypes, the other four comparisons find significantly higher performance for the more complex team structure.

Another perspective on team performance is to consider how effectively the team improves upon the initial soloist solution. We use *normalized improvement* to measure this. Normalized improvement is the fraction of the available improvement between the initial soloist solution



Rich iterative archetype

Figure 5.12: Iterative version of the rich archetype of *Foldit* team structure. Each node corresponds to a team member, with arrows indicating when a solution shared by one member is evolved by another. Red rectangles represent the soloist, blue ovals represent evolvers, and blue diamonds represent evolvers who shared their solution. Note how the soloist shares multiple solutions, continuing their participation in the collaboration past their initial shared solution.

and the best overall solution on that puzzle the team achieved. We compute it by taking the improvement in performance from the initial soloist solution to the team's best solution and dividing it by the difference between the initial soloist solution and the best overall solution on that puzzle. Figure 5.14b shows the distributions of normalized improvement for each team structure archetype. Table 5.6 lists the number of teams in each archetype and the median performance and normalized improvement.

Like with performance, more complex team structures perform better on average, but with two key differences. First, the effect sizes of the differences in normalized improvement are greater than those for differences in performance (see Table 5.5 for the results of the same six comparisons). The advantage of more complex team structures grows stronger when we better isolate the team's contribution by considering improvement instead of absolute performance. Second, the iterative versions of one-branch and n-branch have significantly better normalized improvement than their non-iterative counterparts, including a large effect size for the

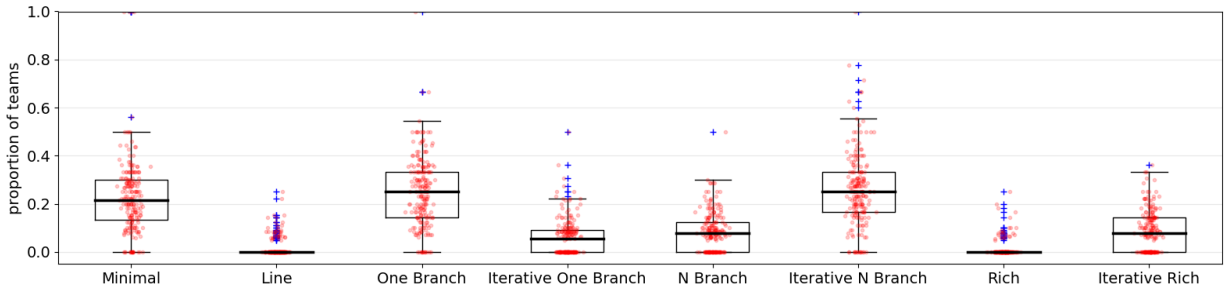
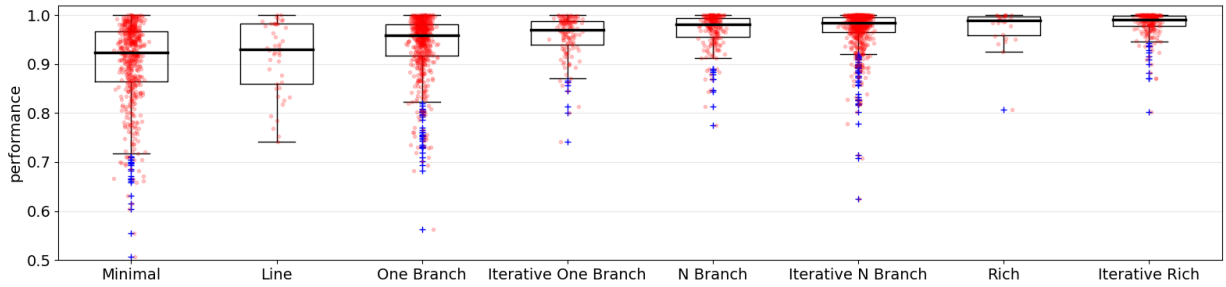
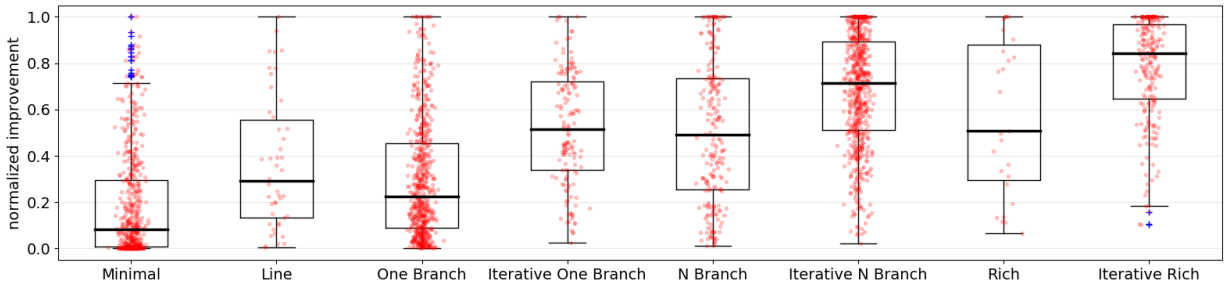


Figure 5.13: The relative occurrence of each team structure archetype across the 209 puzzles in our dataset. For each puzzle, we compute the proportion of teams on that puzzle belonging to each archetype. The boxplots show the distribution of these proportions per archetype, where each red dot is a puzzle. The line and rich archetypes occur rarely, but the populations of the other archetypes are fairly stable.

difference between one-branch and iterative one-branch. Continued engagement from the soloist has a large impact when it comes to realizing the greatest possible improvement. This highlights a powerful potential intervention for a problem-solving system to facilitate the back-and-forth collaboration of the iterative archetypes. There is an important caveat to the generality of this investigation of team structure. While the form of collaboration in *Foldit* maps onto collaborative design or analysis, unlike many domains *Foldit* teams suffer almost no overhead with increased complexity. Extra user effort in the form of sharing and evolving additional solutions increases structural complexity without added costs. Thus, applying these structural insights to other domains is contingent on accounting for overhead.



(a) Distributions of performance



(b) Distributions of normalized improvement

Figure 5.14: The distribution of team performance and normalized improvement across the 209 puzzles in our dataset for each team structure archetype. A team's performance is the quality of the best solution reached by any team member as part of the collaboration. A team's normalized improvement is computed by taking the improvement in performance from the initial soloist solution to the team's best solution and dividing it by the difference between the initial soloist solution and the best overall solution on that puzzle. This metric gets at how much of the available improvement a team was able to realize. There is a clear trend for more complex team structures to perform better and achieve greater improvement.

Comparison	Effect Size
Minimal versus one-branch	-0.301***
One-branch versus n-branch	-0.364***
One-branch versus iterative one-branch	-0.174*
Iterative one-branch versus iterative n-branch	-0.265***
N-branch versus Iterative n-branch	-0.059
Iterative n-branch versus iterative rich	-0.187***

Table 5.4: The results of six comparisons of performance between team structure archetypes in *Foldit*. The rank-biserial correlation measure of effect size is given (negative effect size indicates that the right side of the comparison has higher performance). Significance of the Mann-Whitney U test is indicated by: $^{*}p < 0.004$, $^{**}p < 0.0008$, $^{***}p < 0.00008$. The difference between n-branch and iterative n-branch was not significant.

Comparison	Effect Size
Minimal versus one-branch	-0.354***
One-branch versus n-branch	-0.402***
One-branch versus iterative one-branch	-0.498***
Iterative one-branch versus iterative n-branch	-0.331***
N-branch versus Iterative n-branch	-0.356***
Iterative n-branch versus iterative rich	-0.253***

Table 5.5: The results of six comparisons of normalized improvement between team structure archetypes in *Foldit*. The rank-biserial correlation measure of effect size is given (negative effect size indicates that the right side of the comparison has higher normalized improvement). Significance of the Mann-Whitney U test is indicated by: $^{*}p < 0.004$, $^{**}p < 0.0008$, $^{***}p < 0.00008$.

Archetype	Teams	Performance	Normalized Improvement
Minimal	545	0.924	0.082
Line	45	0.930	0.302
One-branch	619	0.959	0.224
Iterative one-branch	148	0.970	0.514
N-branch	217	0.981	0.491
Iterative n-branch	633	0.984	0.714
Rich	27	0.990	0.509
Iterative rich	219	0.990	0.839

Table 5.6: The number of instances, the median performance, and the median normalized improvement of each team structure archetype in *Foldit*.

5.6 Predicting Team Performance from Team Structure

Just as we explore how factors of group composition, experience, and skill impact group performance in Section 5.4, we also consider how features of team structure relate to team performance. In this section, we delve deeper into *Foldit* team structure to explore what structural properties best predict team performance. We use the same dataset of 2,454 teams from 209 prediction puzzles released between 2015 and 2017. We design eight features of team structure for use in our predictive models, using a team from the dataset visualized in Figure 5.15 as a running example:

Depth: the maximum depth of any contribution to the team. The example team has a depth of six.

Breadth: the maximum number of evolutions at any depth within the team structure. The example team has a breadth of seven (from the branches of four and three occurring at depth five).

Team Size: the number of unique team members. The example team has a size of six.

Number of Evolutions: the number of times a team member evolves a shared solution. The example team has 16 evolutions.

Feedback: the number of times a soloist solution is evolved before the soloist shares another solution (i.e., how many opportunities for feedback via formal evolution did the soloist have as part of the collaboration). The example team has four instances of feedback. Only teams in one of the iterative archetypes will have non-zero feedback.

Start of Collaboration: when during the course of the puzzle did the collaboration start (0 being the moment the puzzle was released and 1 being when the puzzle was closed). The example team started at 0.415.

Connections to Other Teams: we define a *connection* to be when a team member is also a member of another team on the same puzzle. This feature is the number of these connections among all members of the team. The example team has nine connections.

Prior Collaboration: a measure of the amount of experience team members have

collaborating together. We take all subsets of team members of size two or more and compute the number of times n that the members in each subset have been team members on previous puzzles. We then sum $\frac{k(k-1)}{2}n$ for all subsets where k is the size of the subset. In this way we account for dyads or triads within a larger team that have prior experience working together, as well as exponentially weighting situations where larger subsets have previously worked together. The example team has extensive prior collaboration, with 41 of the 57 subsets of team members having at least one prior collaboration. One particularly experienced dyad worked together on 105 previous puzzles. In total, the example team has a prior collaboration score of 231.

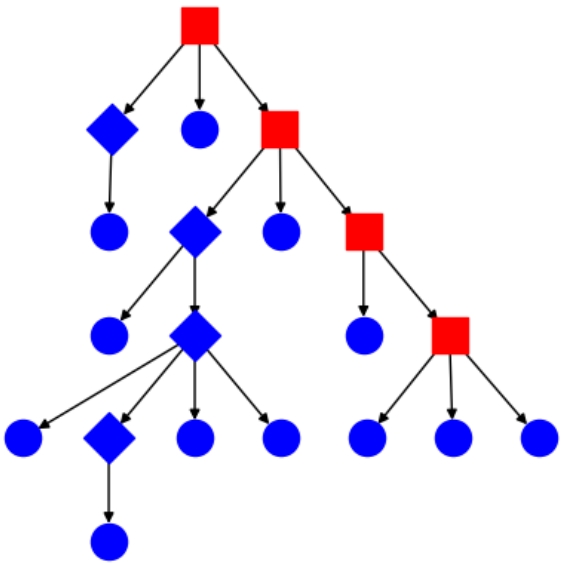


Figure 5.15: An example team structure of the iterative rich archetype. Red squares represent the soloist, blue ovals represent evolvers, and blue diamonds represent evolvers who shared their solution.

We include one additional non-structural feature in our models of team performance: the quality of the initial soloist solution. As *Foldit* collaboration revolves around improving a soloist effort, any sensible model of team performance must account for central role the soloist plays. With these nine features, we use Gradient Boosted Regression Trees (GBRT) [58] to

predict team performance and team normalized improvement. The resulting models have a cross-validated R^2 of 0.588 and 0.475, respectively.

Figure 5.16 shows the relative feature importances for predicting performance. As expected, the quality of the initial soloist solution matters a great deal. This highlights the importance of training and augmenting individuals even in a collaborative context. We observe that depth is many times more important than breadth. This suggests that focused refinement of the current solution leads to better performance than many people working in parallel from a shared starting point. We also observe that the importance of the number of evolutions far outweighs the importance of team size. From this we conclude increased team size is of little value except when it increases evolving activity. Getting more work out of existing team members would likely have a similar effect.

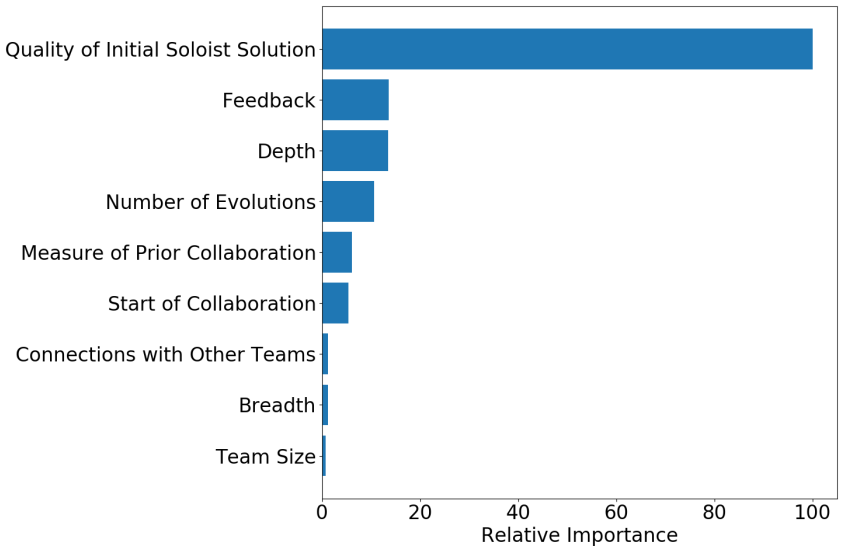


Figure 5.16: The relative feature importances for a model predicting team performance from features of team structure. The value for most important feature is set to 100, and the remaining features are scaled accordingly.

In addition to feature importance, GBRT models can also provide the partial dependence of the prediction on individual features. This isolates the relationship of the prediction to a

single feature by marginalizing over all other features. The value of a partial dependence plot is in illuminating the nature of the relationship between the feature and the prediction. The partial dependence plot in Figure 5.17 shows the strong linear relationship the quality of the initial soloist solution has with team performance. This relationship shows that collaboration is not rescuing mediocre solutions in *Foldit*. Teams appear to generally make incremental refinement rather than transformative improvement.

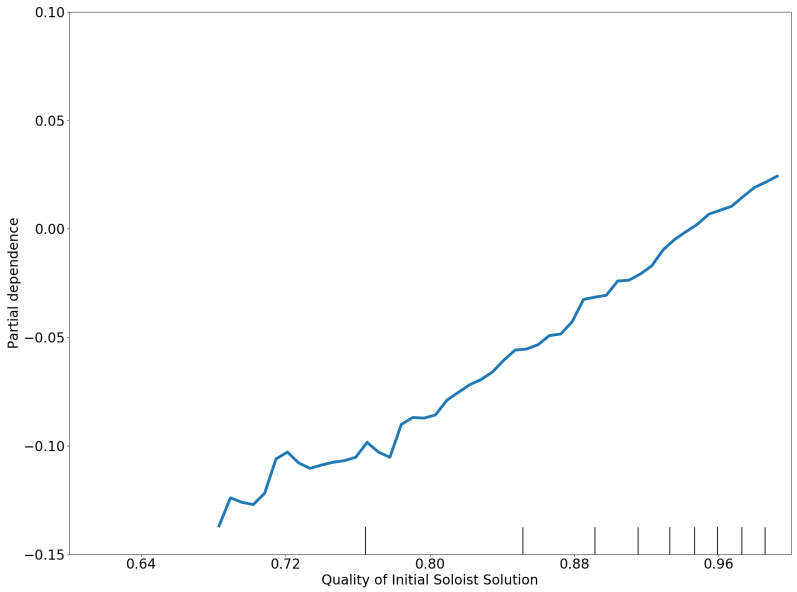


Figure 5.17: The partial dependence of a model predicting team performance on the quality of the initial soloist solution. The marks just above the x-axis are a rug plot showing the quantiles of the underlying data. We observe a strong linear relationship of team performance with initial solution quality.

The partial dependence plots for the eight structural features are shown in Figure 5.18. We can see that feedback and prior collaboration have highly discontinuous relationships with performance. The presence of any feedback corresponds to an iterative archetype, again indicating that this may be a collaborative behavior worth incentivizing or facilitating. Some prior collaboration appears beneficial, but there comes a point where it no longer has any effect. Increased depth and number of evolutions also have diminishing returns, but without

the sharp discontinuity of prior collaboration. Finally, the plot for the start of collaboration indicates a benefit to starting very early and a downside to starting very late, but otherwise variation in when collaboration starts appears to have little impact.

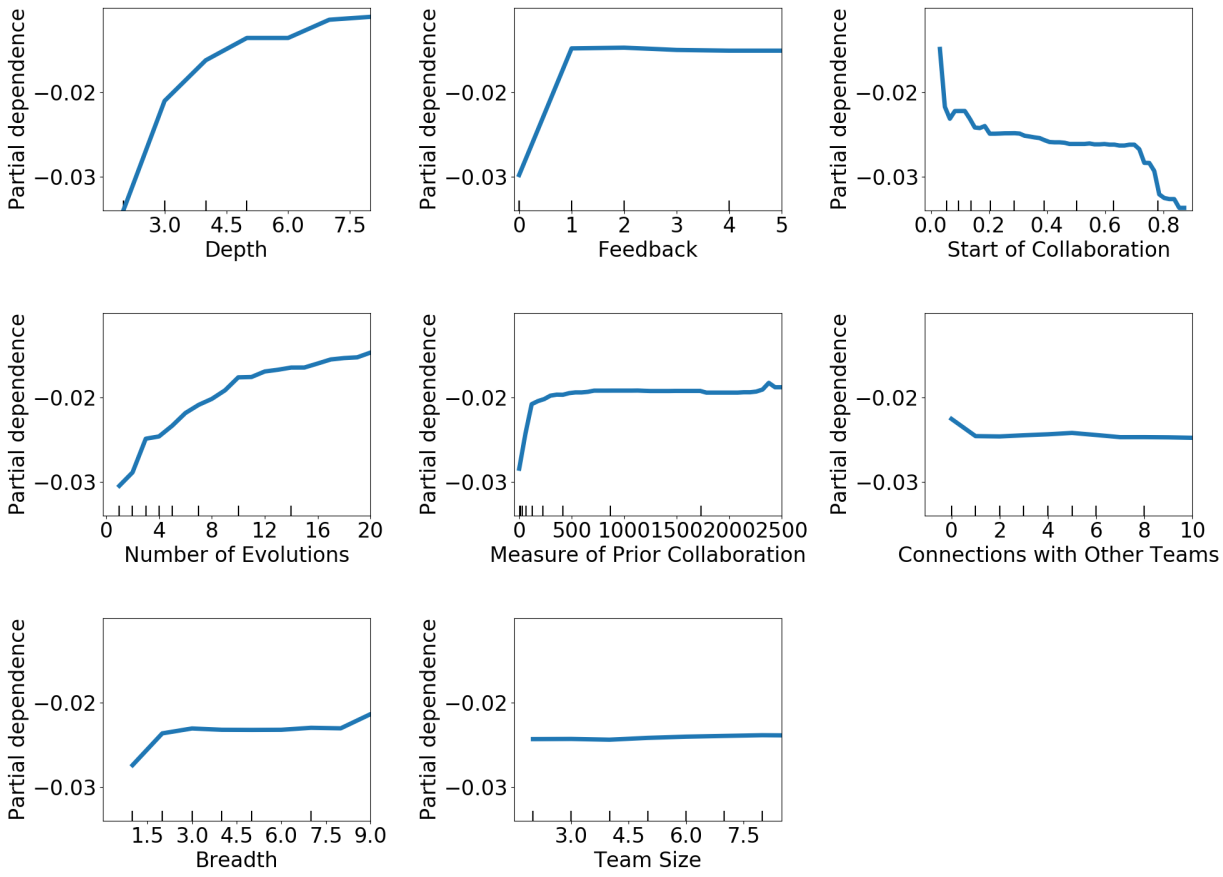


Figure 5.18: The partial dependence of a model predicting team performance on our eight features of team structure. The marks just above the x-axes are rug plots showing the quantiles of the underlying data.

The results for predicting normalized improvement hammer home the importance of iterative refinement to successful collaboration in *Foldit*. As Figure 5.19 shows, depth supplants quality of initial solution as far and away the most important feature. Feedback moves up to the second-most important, though a similar discontinuity seen in Figure 5.21

suggests it remains simply an indication of the benefit of the iterative archetype. The relationship of the quality of the initial soloist solution to normalized improvement is very different than with performance (see Figure 5.20). Solutions on the lower end and very high quality solutions appear most amenable to collaborative improvement. Those in between appear less so, perhaps because they are often mature, sub-optimal solutions, and hence more challenging to improve. The start of collaboration increases in importance for predicting normalized improvement, which makes sense as it directly affects how much time remains in which to make improvement.

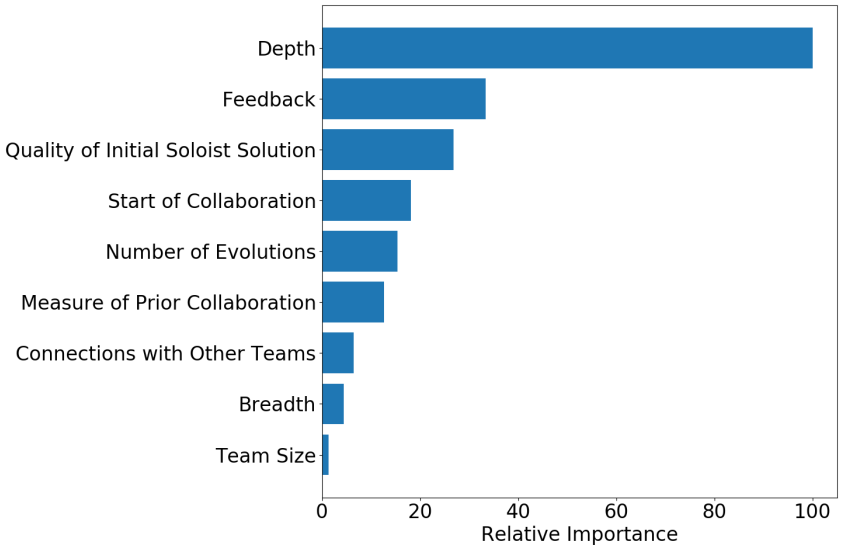


Figure 5.19: The relative feature importances for a model predicting team normalized improvement from features of team structure. The value for most important feature is set to 100, and the remaining features are scaled accordingly.

Prior work identifies team heterogeneity [76] and diversity [8] as beneficial to team performance, though with varying magnitude of effect. We believe it is possible that the importance of iterative refinement (as indicated by the depth, feedback, and number of evolutions features) in our results align with these findings. This refinement could be a mechanism for integrating diverse approaches into the collaborative solution as it cycles

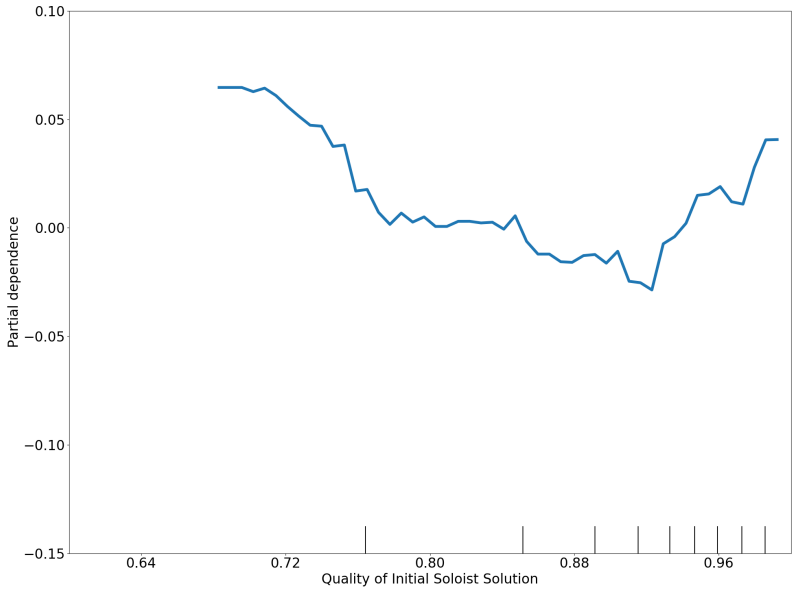


Figure 5.20: The partial dependence of a model predicting team normalized improvement on the quality of the initial soloist solution. The marks just above the x-axis are a rug plot showing the quantiles of the underlying data.

though the hands of multiple team members. Applying the pattern extraction technique presented in Chapter 3 to collaborative work could provide further insight into how the benefits of team diversity extend to an environment like *Foldit*.

In a meta-analysis of the management literature on the relationship of team design features (i.e., how teams are constructed) and team performance, Stewart finds support for positive impacts from increased team size and intrateam coordination [76]. Our results drill down and highlight the specific structural ways larger and more coordinated teams may perform better. Namely, adding depth to the collaboration offers far more benefit than adding breadth. This suggests a role for a problem-solving system to nudge or even actively coordinate users to collaborate in the most effective ways. Intrateam coordination may become stronger as team members gain experience working together. Pobiedina et al. find that playing with friends was the top factor for team success in the competitive online game Dota 2 [60]. While the benefits we observe for prior collaboration support the role of preexisting relationships in

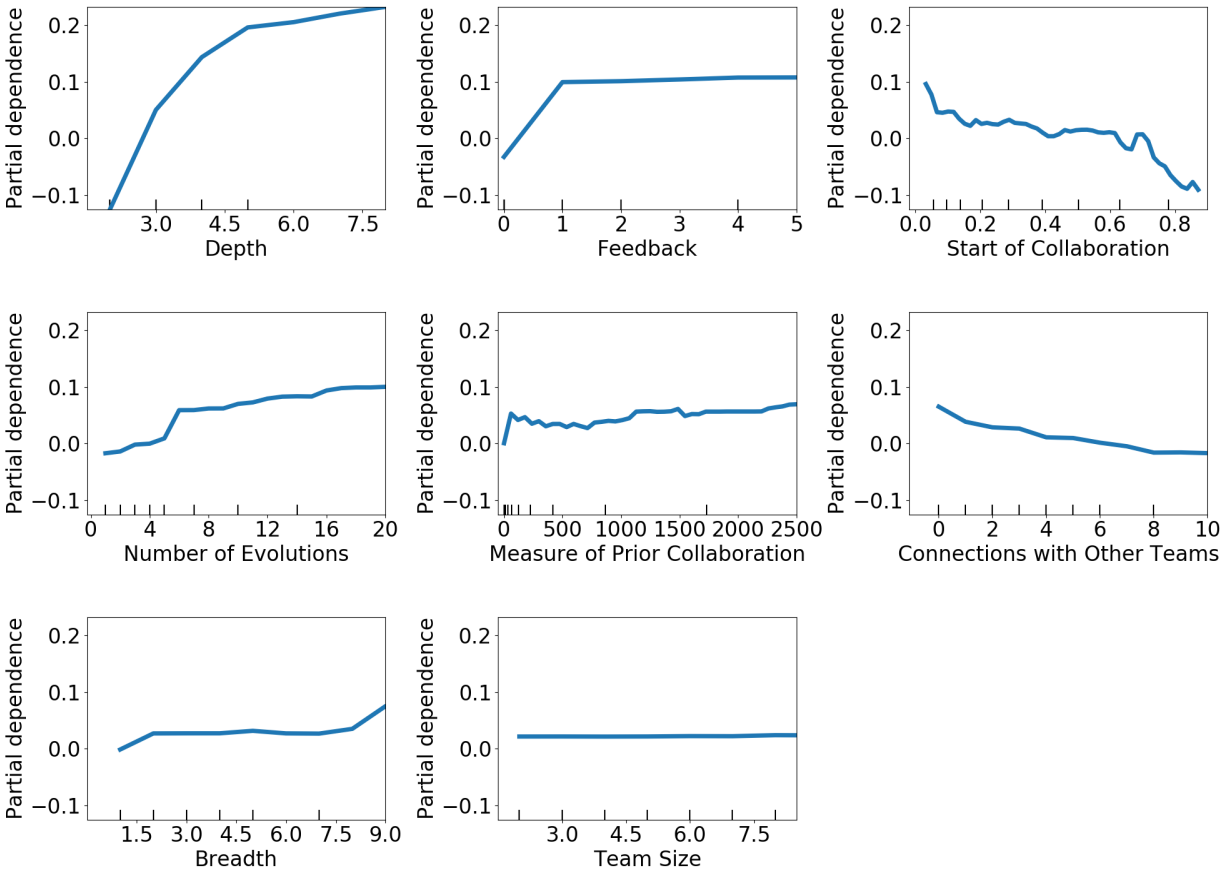


Figure 5.21: The partial dependence of a model predicting team normalized improvement on our eight features of team structure. The marks just above the x-axes are rug plots showing the quantiles of the underlying data.

boosting team performance, they are far from the most important factor for predicting team performance in *Foldit*. It is possible the formal structure of the soloist-evolver model reduces the importance of closely integrated collaboration with familiar actors.

Recent work by Wu et al. studying the relationship of team size with the nature of the team's contribution robustly demonstrates that larger teams tend towards development, while smaller teams tend toward disruption [92] (where more disruptive contributions transform the status quo and supplant prior work and more developmental contributions refine and extend

existing work). As collaboration in *Foldit* is focused on development (i.e., teams try to refine an initial soloist solution), this finding suggests that larger teams would be better suited to the task. By taking the specific structure of the collaboration into account, we show that increased depth of team structure may be the specific value of larger teams in this context. Further analysis of the interaction between team size and other team structure properties is needed to more fully explore how our results connect with other research on team size.

The *DreamTeam* system developed by Zhou et al. is particularly relevant to an empirical discussion of how team structure affects team performance [95]. The researchers recruit teams of three from Amazon Mechanical Turk (AMT) to play a word-guessing game based on the board game *Codenames* and for some of them the *DreamTeam* system uses multi-armed bandits with temporal constraints to discover the optimal settings for five dimensions of team structure. These dimensions are more varied than those included in our model, and three of them focus on norms of social interaction. This makes the *DreamTeam* experiment a valuable complement to our work, as many social aspects of collaboration are not part of the *Foldit* dataset. The authors list extending the team structure dimensions targeted by the system as a primary direction for future work, and analysis like ours provides evidence of possibly impactful additional dimensions. The significant performance benefit for teams whose structure is guided by the *DreamTeam* system over teams in other conditions is a very promising result for the potential of intelligent problem-solving systems to improve collaboration. A key open question is how these kind of interventions would function with real-world teams. The authors report that teams generally responded to interventions altering team structure, but existing teams or an existing community might prove less malleable than AMT workers recruited for a short-term research study. It will be critical to establish whether the benefits of these interventions transfers to real-world user-driven teams in a mature community. In particular, further study is needed to determine if the same benefits could be achieved with lighter-touch interventions or changes in incentives.

Chapter 6

CONCLUSION

The work presented in this dissertation is motivated by the potential of intelligent problem-solving systems to improve and coordinate their users. As the example in Figure 1.2 illustrates, these systems could intervene to accelerate users' transition from novice to expert, to provide guidance and correct mistakes, to bring users together in mutually beneficial arrangements, and to form teams with the right mix of specialties and expertise. These interventions will depend on detailed and reliable measures of user skill, and a deep understanding of problem-solving behavior in the domain of interest. This dissertation makes contributions in both diagnosing individual behavior and understanding collaborative dynamics.

These contributions all relate to my central question: *what makes group and individuals successful problem solvers in open-ended environments?* I have developed new techniques and performed a variety of analyses on the unique open-ended problem-solving domain *Foldit*, as well as the neuron reconstruction game *Mozak* and real-time strategy game *Starcraft II*, to address this question. First, I presented *pattern extraction*, a general technique for automatically extracting patterns from user actions. This included a framework for assembling multivariate time series from logs of user behavior, recursively clustering those time series, and selecting a clustering model to produce the most understandable set of patterns. I evaluated pattern extraction on three problem-solving domains and demonstrated its generality and capacity to produce coherent patterns of behavior. In particular, I used extracted patterns to visualize the surprising diversity of behavior among top users, and identified patterns associated with increased performance beyond the benefits of generic additional effort. Pattern extraction is a general method for discovering domain-specific problem-solving patterns.

In contrast, my next contribution used a domain-specific method to identify more general

structural patterns of problem-solving behavior. Specifically, I used visualization to identify six structural patterns of problem-solving behavior in *Foldit*. I used an iterative methodology to design a visualization of users' problem-solving activity as solution trees, leveraging inherent structure in the problem-solving process not included in the action series used by pattern extraction. This inherent structure is broadly applicable to design and analysis tasks. The size and complexity of the *Foldit* data required me to develop domain-specific techniques to summarize the solution trees and render them tractable for analysis while preserving the salient problem-solving behaviors. I compared the occurrence of the structural patterns I identified between high- and lower-performing users. I found that high-performing users explore more broadly and more aggressively avoid local optima. I also found that both categories of users employ automation and manual intervention in similar quantities.

While my first two contributions are focused on diagnosing individual problem-solving success, my third contribution is a broad-ranging investigation of collaboration in *Foldit*. This investigation was guided by three key questions: (1) how do the social aspects of *Foldit* impact an individual's behavior? (2) what factors have significant impact on group success? and (3) how do users structure their collaboration in *Foldit*? I addressed (1) by analyzing the effects of early collaboration and early competitive success on a user's continued participation and by conducting a synthetic controlled experiment to determine how joining a group affects individual performance. I found that early interaction with each social system, competition and collaboration, is associated with increased performance. I also found that joining a group leads to increased individual performance. To address (2), I analyzed factors affecting both group and team success. For the former, I designed macro- and meso-scale features of group skill, experience, and participation, and found that measures of skill correlate strongly with performance, that participation correlates moderately, and that experience correlates only weakly. For the latter, I designed features of team structure and used Gradient Boosted Regression Trees to examine their relationship to and predictive power for team performance. This examination showed that the amount of collaborative refinement matters far more than parallel exploration or a larger team size. Finally, I addressed (3) by producing an ontology of

eight team structure archetypes. My analysis showed that the population of these archetypes is fairly stable, and that more complex team structures have higher average performance.

While these three contributions stand on their own, I believe there is significant potential in integrating them together. First, the diagnosis of individuals can be applied to the collaborative context. The strong relationship of aggregated team characteristics with team performance identified in prior work (e.g., [76]) argues forcefully for bringing my analysis of individual capabilities to bear on collaboration. The success of real-world teams can often hinge on how effectively the capabilities of individual team members complement one another, and the patterns of problem-solving behavior I am able to identify could prove useful in evaluating team composition. Second, the different approaches I take to identifying individual behaviors are potentially highly complementary. Incorporating pattern extraction results into solution tree visualizations could illuminate sophisticated patterns of exploration not apparent in the current visualizations. Incorporating the structure of solution trees into pattern extraction as signals within the action series could enable patterns to be differentiated according to the structural role they play in the problem-solving process.

Two key extensions of my work could greatly enhance its depth and breadth. My current techniques and analysis are squarely focused on the diagnostic piece of an intelligent problem-solving system. While valuable in its own right, diagnosis also forms the foundation for active intervention. Many of my findings indicate potentially impactful points for intervention in both *Foldit* and *Mozak*. Implementation and evaluation of such interventions would be a powerful demonstration and would give my work added depth. More comprehensive applications of my work to domains outside of *Foldit* would enhance its breadth. Though *Foldit* is a one-of-a-kind system in many respects, it is not the only exciting open-ended problem-solving environment. Online games such as *Starcraft II* are a particularly promising area for future work due to their complexity and the amount of data available.

That said, I am far from exhausting the incredibly unique opportunity *Foldit* and *Mozak* present to study problem solving in an open-ended environment. Expert feedback is a key mechanism in both systems not included in the work presented here. In *Mozak*, professional

neuroscientists monitor the progress of reconstruction efforts and embed comments within the game calling users' attention to mistakes, challenges, or areas in need of attention. In *Foldit*, expert feedback comes into play for design puzzles, where biochemists select a small number of the solutions to try and synthesize in the lab. Experts can also impose additional constraints on future design puzzles to try and guide solutions toward more promising designs. The optimal ways of integrating these kinds of feedback into a problem-solving environment are not well known, and studying them forms an important piece of continued work toward realizing the full potential of intelligent problem-solving systems.

In addition to design puzzles, other interesting variants of *Foldit* puzzles are not included in my analysis. *Hand-folding* puzzles take place over two rounds where the first round does not allow collaboration or recipes and solvers can import first-round solutions into the second round. These puzzles could provide a semi-controlled setting in which to study the effects of collaboration. *All-hands* puzzles treat the entire population of *Foldit* solvers as one big group (i.e., a shared soloist solution is shared with *everyone*). This alternative structure for collaboration could provide an illuminating contrast to the typical one where solvers are siloed into groups.

In this dissertation, my diagnosis (in terms of identifying patterns and predicting performance) of individual and collaborative problem solving is static. This ignores, however, evolution of groups and individuals over time. In extending analysis of problem solving to challenging, new, open-ended domains, I necessarily focused on the simpler, static picture, but understanding how behavior and success change over time will be an essential piece of the full picture. The work presented here is well positioned for this extension: the patterns and features I use to understand behavior in the moment translate naturally to examining how behavior evolves. *Foldit* is a particularly promising domain in which to study this questions, as many users have been dedicated members of the community for years. Adaptation is another dynamic aspect of behavior deserving of future study. Within any problem-solving domain, especially an open-ended one, specific tasks will vary in difficulty, emphasis, and requirements. How users and teams respond to this variation likely plays an important role in

long-term success. A complete picture of problem-solving behavior must account for effective and ineffective adaptation.

BIBLIOGRAPHY

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information Systems*, 53:16–38, 2015.

[2] Erik Andersen, Yun-En Liu, Ethan Apter, François Boucher-Genesse, and Zoran Popović. Gameplay analysis through state projection. In *Proceedings of the 5th International Conference on the Foundations of Digital Games*, pages 1–8. ACM, 2010.

[3] Brigid Barron. Achieving coordination in collaborative problem-solving groups. *The Journal of the Learning Sciences*, 9(4):403–436, 2000.

[4] Blizzard Entertainment. Starcraft 2, 2010.

[5] Mina Shirvani Boroujeni and Pierre Dillenbourg. Discovery and temporal analysis of latent study patterns in mooc interaction sequences. In *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*, pages 206–215. ACM, 2018.

[6] Joel Chan, Steven Dang, and Steven P Dow. Improving crowd innovation with expert facilitation. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, pages 1223–1235. ACM, 2016.

[7] Weiyu Chen, Andrew S Lan, Da Cao, Christopher Brinton, and Mung Chiang. Behavioral analysis at scale: Learning course prerequisite structures from learner clickstreams. *International Educational Data Mining Society*, 2018.

[8] Ziqiang Cheng, Yang Yang, Chenhao Tan, Denny Cheng, Alex Cheng, and Yueting Zhuang. What makes a good team? a large-scale study on the effect of team composition in honor of kings. In *The World Wide Web Conference*, WWW '19, pages 2666–2672, New York, NY, USA, 2019. ACM.

[9] M T H Chi, R Glaser, and E Rees. Expertise in problem solving. ed. rj sternberg, 7-75. In R J Sternberg, editor, *Advances in the Psychology of Human Intelligence*. Lawrence Erlbaum Associates, 1982.

[10] Analía Cicchinelli, Eduardo Veas, Abelardo Pardo, Viktoria Pammer-Schindler, Angela Fessl, Carla Barreiros, and Stefanie Lindstädt. Finding traces of self-regulated learning in activity streams. In *Proceedings of the 8th International Conference on Learning Analytics and Knowledge*, pages 191–200. ACM, 2018.

[11] Seth Cooper, Firas Khatib, Ilya Makedon, Hao Lu, Janos Barbero, David Baker, James Fogarty, Zoran Popović, et al. Analysis of social gameplay macros in the foldit cookbook. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 9–14. ACM, 2011.

[12] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, et al. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.

[13] Dan Cosley, Dan Frankowski, Loren Terveen, and John Riedl. Suggestbot: using intelligent task routing to help people find work in wikipedia. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, pages 32–41. ACM, 2007.

[14] National Research Council et al. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*. National Academies Press, 2000.

[15] Justin Cranshaw and Aniket Kittur. The polymath project: lessons from a successful online collaboration in mathematics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1865–1874. ACM, 2011.

[16] Lex Donaldson. The normal science of structural contingency theory. *Studying Organizations: Theory and Method. Thousand Oaks, Calif: Sage*, pages 51–70, 1999.

[17] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. Shepherding the crowd yields better work. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 1013–1022. ACM, 2012.

[18] Michael Eagle, Drew Hicks, Barry Peddycord III, and Tiffany Barnes. Exploring networks of problem-solving interactions. In *Proceedings of the 5th Conference on Learning Analytics And Knowledge*. ACM, 2015.

[19] Christopher B Eiben, Justin B Siegel, Jacob B Bale, Seth Cooper, Firas Khatib, Betty W Shen, Barry L Stoddard, Zoran Popovic, and David Baker. Increased diels-alderase activity through backbone remodeling guided by foldit players. *Nature Biotechnology*, 30(2):190–192, 2012.

[20] David Engel, Anita Williams Woolley, Ishani Aggarwal, Christopher F Chabris, Masamichi Takahashi, Keiichi Nemoto, Carolin Kaiser, Young Ji Kim, and Thomas W Malone. Collective intelligence in computer-mediated collaboration emerges in different contexts and cultures. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3769–3778. ACM, 2015.

[21] Mohammad H Falakmasir, José P Gonzalez-Brenes, Geoffrey J Gordon, and Kristen E DiCerbo. A data-driven approach for inferring student proficiency from game activity logs. In *Proceedings of the Third (2016) ACM Conference on Learning@Scale*, pages 341–349. ACM, 2016.

[22] Sandra B Fan, Tyler Robison, and Steven L Tanimoto. Cosolve: A system for engaging users in computer-supported collaborative problem solving. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 205–212. IEEE, 2012.

[23] Katherine Fu, Jonathan Cagan, and Kenneth Kotovsky. Design team convergence: the influence of example solution quality. *Journal of Mechanical Design*, 132(11):111005, 2010.

[24] Chase Geigle, ChengXiang Zhai, and Duncan C Ferguson. An exploration of automated grading of complex assignments. In *Proceedings of the Third (2016) ACM Conference on Learning@Scale*, pages 351–360. ACM, 2016.

[25] James G Greeno. Natures of problem-solving abilities. *Handbook of Learning and Cognitive Processes*, 5:239–270, 1978.

[26] J Richard Hackman. *Leading Teams: Setting the Stage for Great Performances*. Harvard Business Press, 2002.

[27] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. Toeplitz inverse covariance-based clustering of multivariate time series data. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 215–223. ACM, 2017.

[28] Erik Harpstead, Christopher J MacLellan, Kenneth R Koedinger, Vincent Aleven, Steven P Dow, and Brad Myers. Investigating the solution space of an open-ended educational game using conceptual feature extraction. In *Proceedings of The 6th Conference on Educational Data Mining*, 2013.

[29] Britton Horn, Amy K Hoover, Jackie Barnes, Yetunde Folajimi, Gillian Smith, and Casper Harteveld. Opening the black box of play: Strategy analysis of an educational game. In *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, pages 142–153. ACM, 2016.

[30] Woei Hung, David H Jonassen, Rude Liu, et al. Problem-based learning. *Handbook of Research on Educational Communications and Technology*, 3:485–506, 2008.

[31] Joshua Introne, Robert Laubacher, Gary Olson, and Thomas Malone. The climate colab: Large scale model-based collaborative planning. In *International Conference on Collaboration Technologies and Systems (CTS)*, pages 40–47. IEEE, 2011.

[32] Alan Irwin. *Citizen science: A Study of People, Expertise and Sustainable Development*. Psychology Press, 1995.

[33] Robin Jeffries, Althea A Turner, Peter G Polson, and Michael E Atwood. The processes involved in designing software. In J R Anderson, editor, *Cognitive Skills and Their Acquisition*, pages 255–283. L. Erlbaum Associates, 1981.

[34] Nicholas R Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.

[35] Matthew Johnson, Michael Eagle, and Tiffany Barnes. Invis: An interactive visualization tool for exploring interaction networks. In *Proceedings of the 6th Conference on Educational Data Mining*, 2013.

[36] David H Jonassen. Instructional design models for well-structured and iii-structured problem-solving learning outcomes. *Educational Technology Research and Development*, 45(1):65–94, 1997.

[37] David H. Jonassen. Toward a Design Theory of Problem Solving. *Educational Technology Research and Development*, 48(4):63–85, 2000.

[38] David A Joyner. Expert evaluation of 300 projects per day. In *Proceedings of the Third (2016) ACM Conference on Learning@Scale*, pages 121–124. ACM, 2016.

[39] Firas Khatib, Seth Cooper, Michael D Tyka, Kefan Xu, Ilya Makedon, Zoran Popović, and David Baker. Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences*, 108(47):18949–18953, 2011.

[40] Firas Khatib, Frank DiMaio, Seth Cooper, Maciej Kazmierczyk, Miroslaw Gilski, Szymon Krzywda, Helena Zabranska, Iva Pichova, James Thompson, Zoran Popović, et al. Crystal structure of a monomeric retroviral protease solved by protein folding game players. *Nature Structural & Molecular Biology*, 18(10):1175–1177, 2011.

[41] Sunyoung Kim, Christine Robson, Thomas Zimmerman, Jeffrey Pierce, and Eben M Haber. Creek watch: pairing usefulness and usability for successful citizen science. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2125–2134. ACM, 2011.

[42] Young Ji Kim, David Engel, Anita Williams Woolley, Jeffrey Yu-Ting Lin, Naomi McArthur, and Thomas W Malone. What makes a strong team?: Using collective intelligence to predict team performance in league of legends. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 2316–2329. ACM, 2017.

[43] John S Kinnebrew, Kirk M Loretz, and Gautam Biswas. A contextualized, differential sequence mining method to derive students' learning behavior patterns. *JEDM— Journal of Educational Data Mining*, 5(1):190–219, 2013.

[44] Aniket Kittur and Robert E Kraut. Harnessing the wisdom of crowds in wikipedia: quality through coordination. In *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, pages 37–46. ACM, 2008.

[45] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 43–52. ACM, 2011.

[46] Mirjam Köck and Alexandros Paramythis. Activity sequence modelling and dynamic clustering for personalized e-learning. *User Modeling and User-Adapted Interaction*, 21(1-2):51–97, 2011.

[47] Kurt Koffka. *Principles of Gestalt Psychology*. Routledge, 1935.

[48] Jill H Larkin, John McDermott, Dorothea P Simon, and Herbert A Simon. Models of competence in solving physics problems. *Cognitive Science*, 4(4):317–345, 1980.

[49] Yun-En Liu, Erik Andersen, Richard Snider, Seth Cooper, and Zoran Popović. Feature-based projections for effective playtrace analysis. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 69–76. ACM, 2011.

[50] Raddick M Jordan, Bracey Georgia, and Gay Pamela. Galaxy zoo: Exploring the motivations of citizen science volunteers. *Astronomy Education Review*, 2010.

[51] Roberto Martinez Maldonado, Kalina Yacef, Judy Kay, Ahmed Kharrufa, and Ammar Al-Qaraghuli. Analysing frequent sequential patterns of collaborative learning activity around an interactive tabletop. In *Educational Data Mining 2011*, 2010.

[52] Sanna Malinen. Understanding user participation in online communities: A systematic literature review of empirical studies. *Computers in Human Behavior*, 46:228–238, 2015.

[53] Laura Malkiewich, Ryan S Baker, Valerie Shute, Shimin Kai, and Luc Paquette. Classifying behavior to elucidate elegant problem solving in an educational game. In *Proceedings of the 9th Conference on Educational Data Mining*, 2016.

[54] Thomas W Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, 26(1):87–119, 1994.

[55] Richard E Mayer. *Thinking, Problem Solving, Cognition*. WH Freeman/Times Books/Henry Holt & Co, 1992.

[56] Allen Newell, John Calman Shaw, and Herbert A Simon. Elements of a theory of human problem solving. *Psychological Review*, 65(3):151, 1958.

[57] Allen Newell and Herbert Alexander Simon. *Human Problem Solving*. Prentice-Hall, 1972.

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[59] Ruth B Pitt. Development of a general problem-solving schema in adolescence and early adulthood. *Journal of Experimental Psychology: General*, 112(4):547, 1983.

[60] Nataliia Pobiedina, Julia Neidhardt, Maria del Carmen Calatrava Moreno, and Hannes Werthner. Ranking factors of team success. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1185–1194. ACM, 2013.

[61] Yaser Norouzzadeh Ravari, Snader Bakkes, and Pieter Spronck. Starcraft winner prediction. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[62] S Ian Robertson. *Problem Solving: Perspectives from Cognition and Neuroscience*. Psychology Press, 2016.

[63] Tyler Robison. *Opening Up the Collaborative Problem-Solving Process to Solvers*. PhD thesis, University of Washington, 2013.

[64] Jeremy Roschelle and Stephanie D Teasley. The construction of shared knowledge in collaborative problem solving. In *Computer Supported Collaborative Learning*, pages 69–97. Springer, 1995.

[65] Jane Roskams and Zoran Popović. Power to the people: Addressing big data challenges in neuroscience by creating a new cadre of citizen neuroscientists. *Neuron*, 92(3):658–664, 2016.

[66] Dana Rotman, Jenny Preece, Jen Hammock, Kezee Procita, Derek Hansen, Cynthia Parr, Darcy Lewis, and David Jacobs. Dynamic changes in motivation in collaborative citizen-science projects. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 217–226. ACM, 2012.

[67] Elizabeth Rowe, Ryan S Baker, and Jodi Asbell-Clarke. Strategic game moves mediate implicit science learning. In *Proceedings of the 8th Conference on Educational Data Mining*, 2015.

[68] Nikol Rummel and Hans Spada. Learning to collaborate: An instructional approach to promoting collaborative problem solving in computer-mediated settings. *The Journal of the Learning Sciences*, 14(2):201–241, 2005.

[69] Robert Sawyer, Jonathan Rowe, Roger Azevedo, and James Lester. Filtered time series analyses of student problem-solving behaviors in game-based learning. *International Educational Data Mining Society*, 2018.

[70] Burr Settles and Steven Dow. Let's get together: the formation and success of online creative collaborations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2009–2018. ACM, 2013.

[71] Benjamin Shih, Kenneth R Koedinger, and Richard Scheines. Unsupervised discovery of student strategies. In *Educational Data Mining 2010*, 2010.

[72] Herbert A Simon. Information-processing theory of human problem solving. *Handbook of Learning and Cognitive Processes*, 5:271–295, 1978.

[73] Jan D Sinnott. A model for solution of ill-structured problems: Implications for everyday and abstract problem solving. In Jan D Sinnott, editor, *Everyday Problem Solving: Theory and Applications*, pages 72–99. Praeger Publishers, 1989.

[74] Burrhus F Skinner. An operant analysis of problem solving. *Behavioral and Brain Sciences*, 7(4):583–591, 1984.

[75] Mike U Smith. *Toward a Unified Theory of Problem Solving: Views from the Content Domains*. Lawrence Erlbaum Associates, Inc, 1991.

[76] Greg L Stewart. A meta-analytic review of relationships between team design features and team performance. *Journal of Management*, 32(1):29–55, 2006.

[77] Greg L Stewart and Murray R Barrick. Team structure and performance: Assessing the mediating role of intrateam process and the moderating role of task type. *Academy of Management Journal*, 43(2):135–148, 2000.

[78] Brian L Sullivan, Christopher L Wood, Marshall J Iliff, Rick E Bonney, Daniel Fink, and Steve Kelling. ebird: A citizen-based bird observation network in the biological sciences. *Biological Conservation*, 142(10):2282–2292, 2009.

[79] John Sweller, Robert F Mawer, and Mark R Ward. Development of expertise in mathematical problem solving. *Journal of Experimental Psychology: General*, 112(4):639, 1983.

[80] Steven Tang, J Peterson, and Z Pardos. Predictive modelling of student behaviour using granular large-scale action data. *The Handbook of Learning Analytics*, pages 223–233, 2017.

[81] Yla R Tausczik, Aniket Kittur, and Robert E Kraut. Collaborative problem solving: A study of mathoverflow. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 355–367. ACM, 2014.

[82] Joseph J Thompson, Mark R Blair, Lihan Chen, and Andrew J Henrey. Video game telemetry as a critical tool in the study of complex skill learning. *PLoS ONE*, 8(9):e75129, 2013.

[83] Edward L Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, 2(4):i, 1898.

[84] Krisztina Tóth, Heiko Rölke, Samuel Greiff, and Sascha Wüstenberg. Discovering students' complex problem solving strategies in educational assessment. In *Proceedings of the 7th Conference on Educational Data Mining*, 2014.

[85] Jonathan Tudge and Barbara Rogoff. Peer influences on cognitive development: Piagetian and vygotskian perspectives. *Lev Vygotsky: Critical Assessments*, 3:32–56, 1999.

[86] G Wallner and S Kriglstein. Visualization-based analysis of gameplay data–a review of literature. *Entertainment Computing*, 4(3):143–155, 2013.

[87] John B Watson. Is thinking merely action of language mechanisms? *British Journal of Psychology*, 11(1):87–104, 1920.

[88] Max Wertheimer. *Productive Thinking.* Harper, 1959.

[89] Andrea Wiggins and Kevin Crowston. From conservation to crowdsourcing: A typology of citizen science. In *44th Hawaii International Conference on System Sciences (HICSS)*, pages 1–10. IEEE, 2011.

[90] Anita Williams Woolley, Ishani Aggarwal, and Thomas W Malone. Collective intelligence and group performance. *Current Directions in Psychological Science*, 24(6):420–424, 2015.

[91] Anita Williams Woolley, Christopher F Chabris, Alex Pentland, Nada Hashmi, and Thomas W Malone. Evidence for a collective intelligence factor in the performance of human groups. *Science*, 330(6004):686–688, 2010.

[92] Lingfei Wu, Dashun Wang, and James A Evans. Large teams develop and small teams disrupt science and technology. *Nature*, 566(7744):378, 2019.

[93] Stefan Wuchty, Benjamin F Jones, and Brian Uzzi. The increasing dominance of teams in production of knowledge. *Science*, 316(5827):1036–1039, 2007.

[94] Eddie Q Yan, Jeff Huang, and Gifford K Cheung. Masters of control: Behavioral patterns of simultaneous unit group manipulation in starcraft 2. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3711–3720. ACM, 2015.

[95] Sharon Zhou, Melissa Valentine, and Michael S Bernstein. In search of the dream team: temporally constrained multi-armed bandits for identifying effective team structures. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 108. ACM, 2018.

[96] Seyedjamal Zolhavarieh, Saeed Aghabozorgi, and Ying Wah Teh. A review of subsequence time series clustering. *The Scientific World Journal*, 2014, 2014.