# Analysis of problem-solving behavior in open-ended scientific-discovery game challenges

Aaron Bauer
awb@cs.washington.edu

Jeff Flatten
jflat06@cs.washington.edu

Zoran Popović
zoran@cs.washington.edu

Center for Game Science, Computer Science and Engineering
University of Washington
Seattle, WA 98195, USA

## ABSTRACT

Problem-solving skills in creative, open-ended domains are both important and little understood. These domains are generally ill-structured, have extremely large exploration spaces, and require high levels of specialized skill in order to produce quality solutions. We investigate problem-solving behavior in one such domain, the scientific-discovery game *Foldit*. Our goal is to discover differentiating patterns and understand what distinguishes high and low levels of problem-solving skill. To address the challenges posed by the scale, complexity, and ill-structuredness of *Foldit* solver behavior data, we devise an iterative visualization-based methodology and use this methodology to design a concise, meaning-rich visualization of the problem-solving process in *Foldit*. We use this visualization to identify key patterns in problem-solving approaches, and report how these patterns distinguish high-performing solvers in this domain.

## Keywords

Problem Solving; Scientific-Discovery Games; Visualization

## 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 [7, 14]. Existing research supports this move, showing that problem-based learning can enhance students' problem-solving and metacognitive skills [11]. 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 [6, 10]. Our work continues this general line of inquiry by examining creative, problem-solving behavior among players in the scientific-discovery game *Foldit*. By modeling the functions of proteins, the workhorses of living cells, *Foldit* challenges players, hereafter referred to as solvers, to resolve the shape of proteins as a 3D puzzle. These puzzles are completely open and often under-specified, making it a highly suitable setting in which to gain insight into student progress through complex solution spaces. In the *Foldit* scientific-discovery community, the focus is on developing people from novices to experts that are eventually capable of solving protein structure problems that are currently unsolved by the scientific community. In fact, solutions produced in *Foldit* have led to three results published in Nature [3, 5, 16]. *Foldit* is an attractive learning space domain because its solvers are capable of contributing to state-of-the-art biochemistry results, and the vast majority of best performing solvers had no exposure to biochemistry prior to joining *Foldit* community. Hence, solver behavior in *Foldit* represents development of highly effective problem-solving in an open-ended domain over long time horizons. In this work, we identify six strategic patterns employed by *Foldit* solvers and show how these patterns differentiate between successful and less successful solvers. These patterns cover instances where solvers investigate multiple hypotheses, explore more greedily or more inquisitively, try to escape local optima, and make structured use of the manual or automated tools available in *Foldit*.

The aspects of the *Foldit* environment that make it an attractive setting in which to study problem solving also present significant challenges. Problems in *Foldit* share many of the properties Jonassen attributes to *design problems*, which they describe as "among the most complex and ill-structured kinds of problems that are encountered in practice" [13]. These properties include a *vague goal with few constraints* (in *Foldit*, the goal is often entirely open-ended: find a good configuration of the protein), *answers that are neither right or wrong, only better or worse*, and *limited feedback* (in *Foldit*, real-time feedback and solution evaluation are limited to a single numerical score corresponding to the protein's current energy state, and solvers frequently must progress through many low-scoring states to reach a good configuration; more nuanced feedback from biochemists is sometimes available, but on a timescale of weeks). The ill-structured nature of problems posed in *Foldit* 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.

The size and complexity of *Foldit*'s problem space presents another major challenge. Even though the logs of solver interactions consist only of regular snapshots of a solver's current solution (along with attendant metadata), the record of a single solver'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* solvers. In addition to manual manipulation of the protein configuration, solvers can invoke various low-level automated optimization routines (some

of which run until the solver 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. Solvers 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 behavior in *Foldit* intractable.

To overcome these obstacles, we devise a visualization-based methodology capable of producing tractable representations of *Foldit* solvers' problem-solving behavior while maintaining the key encodings necessary for analysis of high-level strategic behavior. 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 solvers' unique strategic behavior. Using this methodology, we examine solver activity logs from 11 *Foldit* puzzles, representing 970 distinct solvers 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 solvers in an open-ended domain like *Foldit*? In addressing these questions we find that high-performing solvers 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 solvers 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.

## 2. RELATED WORK

While automated grading has mostly been explored for well-specified tasks where the correct answer has a straightforward and concise description, some previous work has developed techniques for more complex activities. Some achieve scalability through a crowd-sourcing framework such as Udacity's system for hiring external experts as project reviewers [14]. Other work has demonstrated automated approaches that leverage machine learning to enable scalable grading of more complex assignments. For example, Geigle et al. describe an application of online active learning to minimize the training set a human grader must produce [7] when automatically grading an assignment where students must analyze medical cases. Our work does not focus on grading problem-solving behavior, but instead approaches the issue of scalability at a more fundamental level: understanding fine-grained problem-solving strategies and how they contribute to success in an open-ended domain.

A robust body of prior work has addressed the challenge of both visualizing and gleaning insight from player activity in educational and serious games. Andersen et al. developed Playtracer, a general method for visualizing players' progress through a game's state space when a spatial relationship between the player and the virtual environment is not available [1]. Wallner and Kriglstein provide a thorough review of visualization-based analysis of gameplay data [21]. Prior work has analyzed gameplay data without visualization as well. Falakmasir et al. propose a data analysis pipeline for modeling player behavior in educational games. This system can produce a simple, interpretable model of in-game actions that can predict learning outcomes [6]. Our work differs in its aims from this prior work. We do not seek to develop a general visualization technique, but instead to design and leverage a domain-specific visualization to analyze problem-solving behavior. We are also not predicting player behavior, nor modeling players in terms of low-level actions, but rather identifying higher-level strategy use.

The work most similar to ours is that which focuses on problem-solving behavior, including both the long-running efforts in educational psychology to develop general theories and more recent work data-driven on understanding the problem-solving process. Our formulation of solving behavior in *Foldit* as a search through a problem space follows from classic information-processing theories of problem solving (e.g., [9, 19]). Gick reviews research on both problem-solving strategies and the differences in strategy use between experts and novices [8]. Our work complements the existing literature by focusing on understanding problem solving in the little-studied domain of scientific-discovery games, and on the ill-structured problems present in *Foldit*. Our findings on the differences in strategy use between high- and lower-performing solvers in *Foldit* are consistent with the consensus in the literature that expert's knowledge allows them to effectively use strategies that are poorly or infrequently used by less-skilled solvers. We also contribute a granular understanding of the specific strategies and differences at work in the *Foldit* domain.

Significant recent work has investigated problem-solving behavior in educational games and intelligent tutoring systems using a variety of techniques. Tóth et al. used clustering to characterize problem-solving behavior on tasks related to understanding a system of linear structural equations. The clusters distinguished between students that used a *vary-one-thing-at-a-time* strategy (both more and less efficiently) and those that used other strategies [20]. 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. They found that the use of some moves mediated the relationship between prior achievement and post scores [18]. Eagle et al. discuss several applications of using *interaction networks* to visualize and categorize problem-solving behavior in education games and intelligent tutoring systems. These networks offer insight for hint generation and a flexible method for visualizing student work in rule-using problem solving environments [4] . Using decision trees to build separate models for optimal and non-optimal student performance, Malkiewich et al. gained insight into how learning environments can encourage elegant problem solving [17]. Our primary contribution is to extend analysis of problem-solving behavior to a more complex and open-ended domain that those studied in similar previous work. The size and complexity of *Foldit*'s problem space, the volume of data necessary to capture exploration in this space, and the ill-structured nature of the *Foldit* problems all pose unique challenges. We devise a visualization-based methodology focused on iterative summarization, and successfully apply it to identify key problem-solving patterns exhibited by *Foldit* solvers.

## 3. FOLDIT

*Foldit* is a scientific-discovery game that crowdsources protein folding. It presents solvers with a 3D representation of a protein and tasks them with manipulating it into the lowest energy configuration. Each protein posed to the solvers is called a puzzle. Solvers' solutions to each puzzle are scored according to their energy configuration, and solvers compete to produce the highest scoring results.

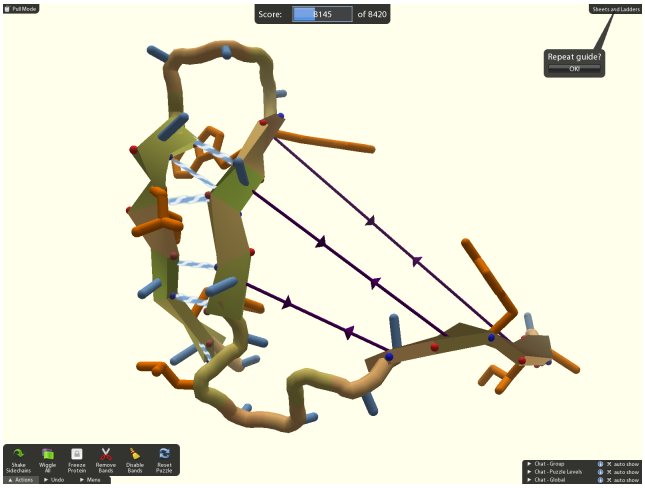


**Figure 1: The *Foldit* interface. *Foldit* solvers use a variety of tools to interactively reshape proteins. In this figure, a solver uses rubber bands to pull together two sheets, long flat regions of the protein.**

Solvers 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. There is, however, a subset of the basic actions that cannot be used by recipes. We will call these *manual-only actions*. Previous work analyzing solver behavior in *Foldit* has focused primarily on recipe use and dissemination [2] and recipe authoring [15].

*Foldit* has several different types of puzzles for solvers to solve. In this work, we 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 solvers 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. We focus on prediction puzzles in this work to simplify our analysis by having a consistent objective (i.e., maximize score) across the problem-solving behavior we analyze.

## 4. METHODOLOGY

Prior work has demonstrated the power of visualization to support understanding of problem-solving behavior (e.g., [12]). 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 solver behavior. We are specifically interested in how solvers navigate from a puzzle's start state to a high-quality solution, what states they pass through in between, and what other avenues they explored. Since solving a *Foldit* puzzle can be represented as a directed search through a problem space, the clear encoding of parent-child relationships between nodes offered by a tree make it well-suited for visualizing these aspects of the solving process.

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 strategic behavior 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 structures corresponding to unique strategic behavior. 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 patterns of strategic behavior among *Foldit* solvers. We follow up on these patterns with computational investigation, and quantify their application by high- and lower-performing solvers.

### 4.1 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 solver's progress through the problem space has only been collected since mid-2015. Our chosen dataset represents 970 unique solvers 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 solver 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 solver 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 solver undoes an action or reloads a previous solution. Hence, by comparing the histories of two snapshots from the same solver, 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 solver's solving process.

## 4.2 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 strategies and approaches. 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 solver's progression. Third, the solver'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 solver solutions, we establish parent-child relationships between solutions. If solution $\beta$ is a child of solution $\alpha$, it indicates that $\beta$ was generated when the solver performed actions on $\alpha$. One crucial limitation, however, is that a snapshot of the solver's current solution is captured far less often (only once every two minutes) than the solver 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 2 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 2. 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.

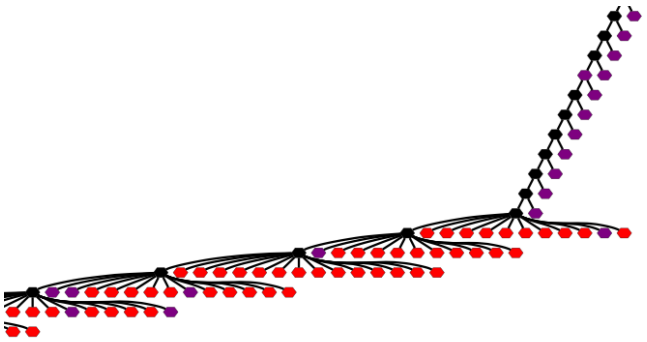

**Figure 2: A solution tree after only the first stage of summarization. The non-black node color represents the score of the solution at that node (red is worse). The black nodes are empty in that we do not have solution data corresponding to that node. This figure also shows examples of the features 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.**

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 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 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 solver 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.

# 5. RESULTS

Using visualized solution trees for a large set of solvers across our sample of 11 puzzles, we identify a set of six prominent patterns in solvers' problem-solving behavior. These patterns do not encompass all solving behavior in *Foldit*, but instead capture key instances of strategic behavior in three categories: exploration, optimization, and human-computer collaboration. Future work is needed to generate a comprehensive survey of the strategic 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 initial

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 solvers to gain an understanding of how these patterns contribute to success in an open-end environment like *Foldit*.

## 5.1 Problem-Solving Patterns

*Exploration.* *Foldit* solvers 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 solvers make significant progress on distinct branches of the tree (see Figure 3 for an example). We interpret this pattern as the solver 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.

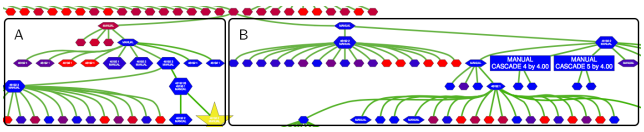


**Figure 3: An example of the *multiple hypotheses* pattern. The two hypotheses branch out one of the nodes at the top and continue to the left (A) and right (B).**

At the micro-scale, solvers 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, solvers 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 shows an example of this behavior.

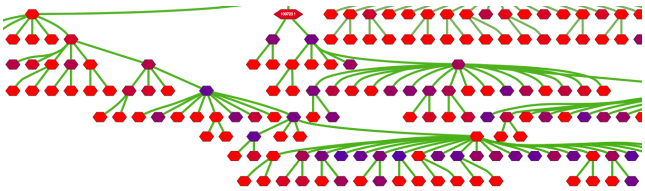

**Figure 4: An example of the *inquisitive* pattern. Note how frequently multiple children of the same node are explored when compared to the tree in Figure 3.**

*Optimization.* Navigating the extremely heterogeneous solution space is the primary challenge in *Foldit*, so we look closely at how solvers attempt to optimize their solutions, digging deeper into solvers' approach to exploration than the previous two patterns. We identify two related patterns describing solvers' fine-grained approach to optimization. The solution spaces of *Foldit* puzzles contain numerous local optima that solvers must escape, and we identify an *optima escape* pattern highly suggestive of a deliberate attempt to escape a local optima. This pattern occurs when a solver has a high-scoring node with a low-scoring child, and then chooses to explore from the low-scoring child. The solver was willing to ignore the short-term drop in score to try and reach a more beneficial state in the long-term. Figure 5 gives an example of this pattern.

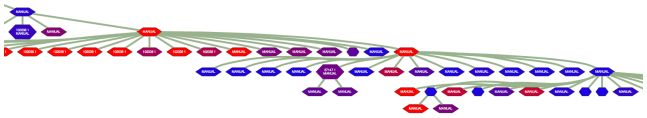

**Figure 5: An example of the *optima escape* pattern. The solver transitions from a relatively high-scoring (i.e., blue) state in the upper left to a low-scoring (i.e., red) state. What makes this an example of the pattern is that exploration from the low-scoring state. In this case, the perseverance paid off as the solver reaches even higher-scoring states in the lower right.**

In the other direction, we identify the *greedy* pattern in which solvers 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 deserves recognition as a pattern with significant potential impact on problem-solving success. Naturally, these two patterns do not cover all the ways solvers explore the problem space, but they do characterize specific strategic behavior of interest in this analysis.

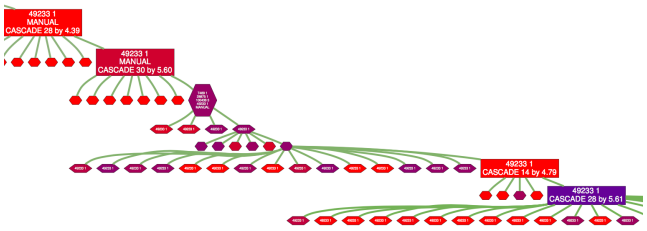


**Figure 6: An example of the *repeated recipe* pattern. At three points in this solution tree snippet, the solver applies recipe 49233 to every child of a node.**

*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 solver 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 solvers 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 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.

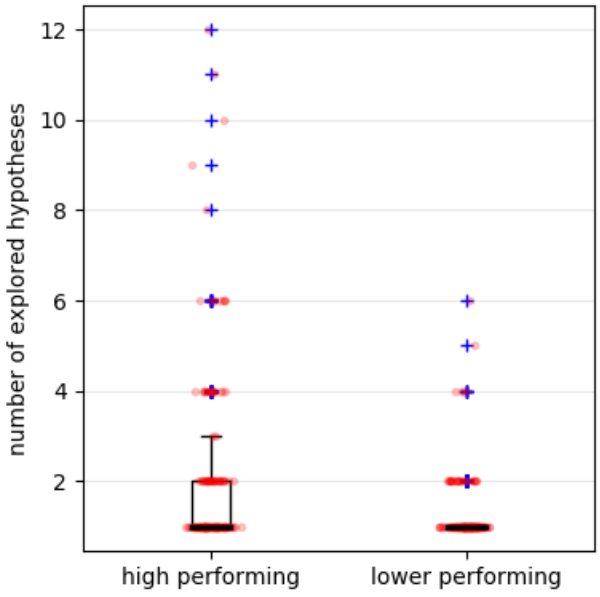


**Figure 7: The number of hypotheses pursued in each solution tree for high- and lower-performing solvers. High-performing solvers frequently pursue two or more hypotheses, whereas lower-performing solvers most often pursue just one. Red circles show the distribution of individual solvers.**

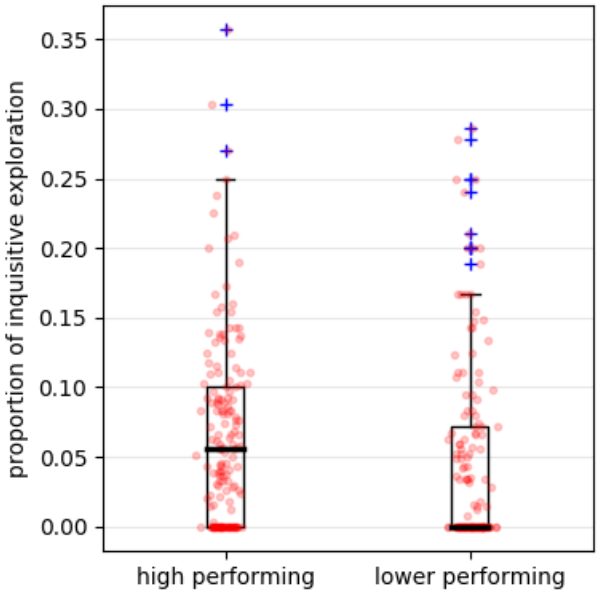


**Figure 8: The proportion of all the branches in a solver's solution tree in which the solver explored more than one child for high- and lower-performing solvers. Red circles show the distribution of individual solvers.**

## 5.2 Problem-Solving Patterns and Solver 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 solvers to that among lower-performing solvers. 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 solvers involved in the comparison $n$ for each test (the full sample is $n = 330$).

We find high-performing solvers explore more broadly than lower-performing solvers. For the *multiple hypotheses* pattern, high-performing solvers pursued significantly more hypotheses than lower-performing solvers ($U = 10569$, $p = 0.000014$, $r = 0.217$, $n = 330$) (see Figure 7). For the *inquisitive* pattern, we compute the proportion of each solver's exploration that matches the pattern (i.e., of all the branches in a solver's solution tree, in what fraction of them did the solver explore more than one child) and find high-performing solvers explore inquisitively more often than lower-performing solvers ($U = 9343$, $p = 0.000295$, $r = 0.231$, $n = 313$) (see Figure 8).

We also find high-performing solvers 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 solvers engage in this behavior more than lower-performing solvers ($U = 11183.5$, $p = 0.00185$, $r = 0.173$, $n = 330$) (see Figure 9). For the *greedy* pattern, we compute the proportion of each solver's exploration that matches the pattern (i.e., of all the branches in a solver's solution tree, in what fraction of them did the solver only explore the best-scoring child). While high-performing solvers engaged in greedy optimization less often than lower-performing solvers, the difference was not significant ($U = 9079$, $p = 0.0158$, $r = -0.163$, $n = 295$) (see Figure 10).

Finally, we find no significant difference between high- and lower-performing solvers 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 solvers ($U = 13334$, $p = 0.789$, $r = 0.014$, $n = 330$). 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 solver trigger at least one recipe) and though lower-performing solvers used recipes more frequently, the difference between high- and lower-performing solvers was not significant ($U = 11342$, $p = 0.0140$, $r = -0.157$, $n = 329$).

# 6. DISCUSSION

The results from our analysis of our solution tree visualizations illuminate some key problem-solving patterns exhibited by individual *Foldit* solvers. Namely, how broadly an individual explores, both on a macro- and micro-scale, how actively an individual avoids

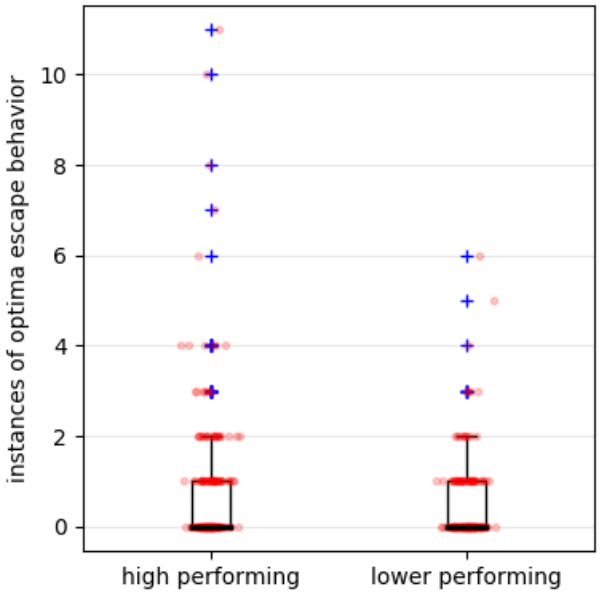


**Figure 9: The number of times in each solution a solver engages in *optima escape* behavior for high- and lower-performing solvers. Red circles show the distribution of individual solvers.**

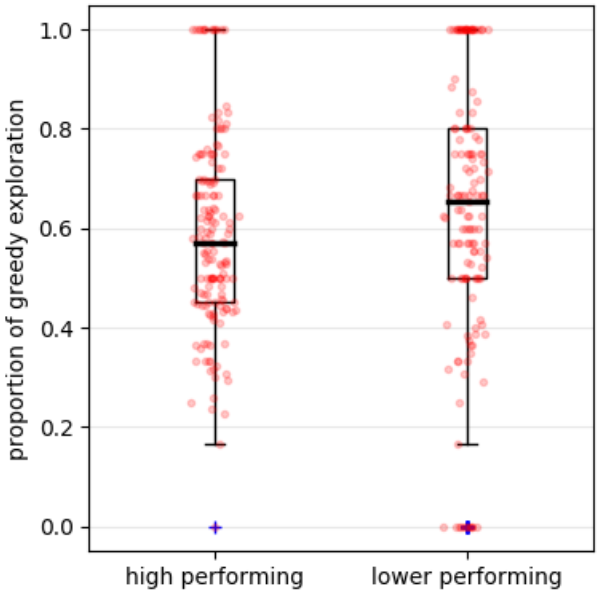


**Figure 10: The proportion of all the branches in a solver's solution tree in which the solver explored only the best-scoring child for high- and lower-performing solvers. The fact that the median for both categories of solver is above 0.5 indicates that this pattern in an important part of refining solutions in *Foldit*. Red circles show the distribution of individual solvers.**

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.

Comparing high- and lower-performing solvers 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 key feature of high-skill solving behaviors is not being enamored by the current best solution and possessing strategies for avoiding myopic thinking had implications for the strategies 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 solvers of different skill 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 behavior, the timing and sequencing of strategic moves are likely to be critical to success. Further work is needed to investigate what differentiates effective and ineffective use of specific solving strategies.

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 solver 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 solvers. 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.

## 7. CONCLUSION

Gaining a better understanding of key patterns in problem-solving behavior in complex, open-ended environments is important for deploying this kind of activity in an educational setting at scale. In this work, we identified six key patterns in problem-solving behavior among solvers of *Foldit*. The protein folding challenges in *Foldit* present rich, completely open, heterogeneous solution spaces, making them a compelling domain in which to analyze these patterns. To facilitate the identification of these patterns, we used an iterative methodology to design visualizations of solvers' problem-solving activity as solution trees. The size and complexity of the *Foldit* data required us to develop domain-specific techniques to summarize the solution trees and render them tractable for analysis while preserving the salient problem-solving behaviors. Finally, we compared the

occurrence of the patterns we identified between high- and lower-performing solvers. We found that high-performing solvers explore more broadly and more aggressively avoid local optima. We also found that both categories of solvers employ automation and manual intervention in similar quantities, inviting future work to study how these tools are used at a more fine-grained level.

We have only scratched the surface in our analysis of a subset of *Foldit* data. Two integral aspects of the *Foldit* environment are not within the scope of this work: collaboration and expert feedback. We only considered solutions produced by individual solvers, but *Foldit* solver can also take solutions produced by others and try and improve them. This collaborative framework may involve specialization and unique solving strategies, and deserves careful study. Expert feedback comes into play for design puzzles, where biochemists will select a small number of the solutions to try and synthesize in the lab. Experts will also impose additional constraints on future design puzzles to try and guide solutions toward more promising designs. The interaction of these channels for expert feedback and problem-solving behavior is an important topic for future research. Also outside the scope of this work is how individual solvers change their problem-solving behavior over time. Many solvers have been participating in the *Foldit* community for many years, and studying how their behavior evolves could yield insights into the acquisition of high-level problem-solving skills.

Looking more broadly at the impact of this work, our methodology and analysis can serve as a first step toward discovering the scaffolding necessary to develop high-level problem-solving skills. These results could contribute to a hint generation system, where solvers could be guided toward known effective strategies, or a meta-planner component in *Foldit* that could tailor the parameters of particular puzzles to optimize the quality of the scientific results. In all of these cases, this work contributes to the necessary foundational understanding of the problem-solving behavior involved.

## 8. ACKNOWLEDGEMENTS

This work was supported by the National Institutes of Health grant 1UH2CA203780, RosettaCommons, and Amazon. This material is based upon work supported by the National Science Foundation under Grant No. 1629879.

## 9. REFERENCES

[1] E. Andersen, Y.-E. Liu, E. Apter, F. Boucher-Genesse, and Z. Popović. Gameplay analysis through state projection. In *Proceedings of the fifth international conference on the foundations of digital games*, pages 1–8. ACM, 2010.

[2] S. Cooper, F. Khatib, I. Makedon, H. Lu, J. Barbero, D. Baker, J. Fogarty, Z. 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.

[3] S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, M. Beenen, A. Leaver-Fay, D. Baker, Z. Popović, et al. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.

[4] M. Eagle, D. Hicks, B. Peddycord III, and T. Barnes. Exploring networks of problem-solving interactions. In *Proceedings of the 5th Conference on Learning Analytics And Knowledge*. ACM, 2015.

[5] C. B. Eiben, J. B. Siegel, J. B. Bale, S. Cooper, F. Khatib, B. W. Shen, B. L. Stoddard, Z. Popovic, and D. Baker. Increased diels-alderase activity through backbone remodeling guided by foldit players. *Nature biotechnology*, 30(2):190–192, 2012.

[6] M. H. Falakmasir, J. P. Gonzalez-Brenes, G. J. Gordon, and K. 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.

[7] C. Geigle, C. Zhai, and D. 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.

[8] M. L. Gick. Problem-solving strategies. *Educational psychologist*, 21(1-2):99–120, 1986.

[9] J. G. Greeno. Natures of problem-solving abilities. *Handbook of learning and cognitive processes*, 5:239–270, 1978.

[10] E. Harpstead, C. J. MacLellan, K. R. Koedinger, V. Aleven, S. P. Dow, and B. 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.

[11] W. Hung, D. H. Jonassen, R. Liu, et al. Problem-based learning. *Handbook of research on educational communications and technology*, 3:485–506, 2008.

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

[13] D. H. Jonassen. Toward a design theory of problem solving. *Educational Technology Research and Development*, 48(4):63–85, dec 2000.

[14] D. 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.

[15] F. Khatib, S. Cooper, M. D. Tyka, K. Xu, I. Makedon, Z. Popović, and D. Baker. Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences*, 108(47):18949–18953, 2011.

[16] F. Khatib, F. DiMaio, S. Cooper, M. Kazmierczyk, M. Gilski, S. Krzywda, H. Zabranska, I. Pichova, J. Thompson, Z. 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.

[17] L. Malkiewich, R. S. Baker, V. Shute, S. Kai, and L. Paquette. Classifying behavior to elucidate elegant problem solving in an educational game. In *Proceedings of the 9th Conference on Educational Data Mining*, 2016.

[18] E. Rowe, R. S. Baker, and J. Asbell-Clarke. Strategic game moves mediate implicit science learning. In *Proceedings of the 8th Conference on Educational Data Mining*, 2015.

[19] H. A. Simon. Information-processing theory of human problem solving. *Handbook of learning and cognitive processes*, 5:271–295, 1978.

[20] K. Tóth, H. Rölke, S. Greiff, and S. Wüstenberg. Discovering students' complex problem solving strategies in educational assessment. In *Proceedings of the 7th Conference on Educational Data Mining*, 2014.

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