Algorithm discovery by protein folding game players

Firas Khatib[a], Seth Cooper[b], Michael D. Tyka[a], Kefan Xu[b], Ilya Makedon[b], Zoran Popović[b], David Baker[a,c,1], and Foldit Players

[a]Department of Biochemistry; [b]Department of Computer Science and Engineering; and [c]Howard Hughes Medical Institute, University of Washington, Box 357370, Seattle, WA 98195

Contributed by David Baker, October 5, 2011 (sent for review June 29, 2011)

Foldit is a multiplayer online game in which players collaborate and compete to create accurate protein structure models. For specific hard problems, Foldit player solutions can in some cases outperform state-of-the-art computational methods. However, very little is known about how collaborative gameplay produces these results and whether Foldit player strategies can be formalized and structured so that they can be used by computers. To determine whether high performing player strategies could be collectively codified, we augmented the Foldit gameplay mechanics with tools for players to encode their folding strategies as "recipes" and to share their recipes with other players, who are able to further modify and redistribute them. Here we describe the rapid social evolution of player-developed folding algorithms that took place in the year following the introduction of these tools. Players developed over 5,400 different recipes, both by creating new algorithms and by modifying and recombining successful recipes developed by other players. The most successful recipes rapidly spread through the Foldit player population, and two of the recipes became particularly dominant. Examination of the algorithms encoded in these two recipes revealed a striking similarity to an unpublished algorithm developed by scientists over the same period. Benchmark calculations show that the new algorithm independently discovered by scientists and by Foldit players outperforms previously published methods. Thus, online scientific game frameworks have the potential not only to solve hard scientific problems, but also to discover and formalize effective new strategies and algorithms.

citizen science | crowd-sourcing | optimization | structure prediction | strategy

Citizen science is an approach to leveraging natural human abilities for scientific purposes. Most such efforts involve visual tasks such as tagging images or locating image features (1–3). In contrast, Foldit is a multiplayer online scientific discovery game, in which players become highly skilled at creating accurate protein structure models through extended game play (4, 5). Foldit recruits online gamers to optimize the computed Rosetta energy using human spatial problem-solving skills. Players manipulate protein structures with a palette of interactive tools and manipulations. Through their interactive exploration Foldit players also utilize user-friendly versions of algorithms from the Rosetta structure prediction methodology (6) such as *wiggle* (gradient-based energy minimization) and *shake* (combinatorial side chain rotamer packing). The potential of gamers to solve more complex scientific problems was recently highlighted by the solution of a long-standing protein structure determination problem by Foldit players (7).

One of the key strengths of game-based human problem exploration is the human ability to search over the space of possible strategies and adapt those strategies to the type of problem and stage of problem solving (5). The variability of tactics and strategies stems from the individuality of each player as well as multiple methods of sharing and evolution within the game (group play, game chat), and outside of the game [wiki pages (8)]. One way to arrive at algorithmic methods underlying successful human Foldit play would be to apply machine learning techniques to the detailed logs of expert Foldit players (9). We chose instead to rely on a superior learning machine: Foldit players themselves. As the players themselves understand their strategies better than anyone, we decided to allow them to codify their algorithms directly, rather than attempting to automatically learn approximations. We augmented standard Foldit play with the ability to create, edit, share, and rate gameplay macros, referred to as "recipes" within the Foldit game (10). In the game each player has their own "cookbook" of such recipes, from which they can invoke a variety of interactive automated strategies. Players can share recipes they write with the rest of the Foldit community or they can choose to keep their creations to themselves.

In this paper we describe the quite unexpected evolution of recipes in the year after they were released, and the striking convergence of this very short evolution on an algorithm very similar to an unpublished algorithm recently developed independently by scientific experts that improves over previous methods.

Results

In the social development environment provided by Foldit, players evolved a wide variety of recipes to codify their diverse strategies to problem solving. During the three and a half month study period (see *Materials and Methods*), 721 Foldit players ran 5,488 unique recipes 158,682 times and 568 players wrote 5,202 recipes. We studied these algorithms and found that they fell into four main categories: (*i*) *perturb and minimize*, (*ii*) *aggressive rebuilding*, (*iii*) *local optimize*, and (*iv*) *set constraints*. The first category goes beyond the deterministic minimize function provided to Foldit players, which has the disadvantage of readily being trapped in local minima, by adding in perturbations to lead the minimizer in different directions (11). The second category uses the *rebuild* tool, which performs fragment insertion with loop closure, to search different areas of conformation space; these recipes are often run for long periods of time as they are designed to rebuild entire regions of a protein rather than just refining them (Fig. S1). The third category of recipes performs local minimizations along the protein backbone in order to improve the Rosetta energy for every segment of a protein. The final category of recipes assigns constraints between beta strands or pairs of residues (*rubber bands*), or changes the secondary structure assignment to guide subsequent optimization.

Different algorithms were used with very different frequencies during the experiment. Some are designated by the authors as public and are available for use by all Foldit players, whereas others are private and available only to their creator or their Foldit team. The distribution of recipe usage among different players is shown in Fig. 1 for the 26 recipes that were run over 1,000 times. Some recipes, such as the one represented by the leftmost bar, were used many times by many different players, while others, such as the one represented by the pink bar in the

Author contributions: F.K., S.C., Z.P., and D.B. designed research; F.K., S.C., M.D.T., and F.P. performed research; F.K., S.C., M.D.T., K.X., and I.M. analyzed data; and F.K., S.C., Z.P., and D.B. wrote the paper.

The authors declare no conflict of interest.

Freely available online through the PNAS open access option.

[1]To whom correspondence should be addressed. E-mail: dabaker@u.washington.edu.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1115898108/-/DCSupplemental.

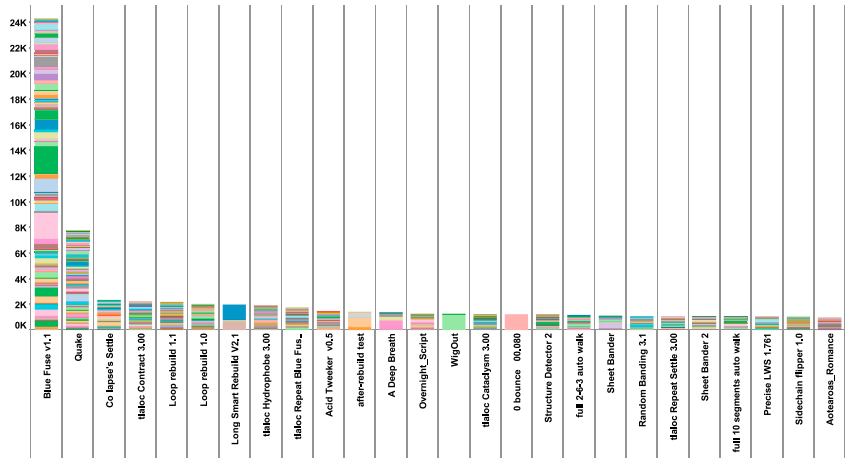


Fig. 1. Foldit recipes are used with very different frequencies. Each bar represents a different Foldit recipe; the height of the bar shows the number of times the recipe was used, and the colors denote different Foldit players. Blue Fuse, at the very left, was run by many players a total of 24,273 times; the creator of Blue Fuse alone (large pink region) used it over 2,000 times. Most recipes (those represented by many colors) were run by different players; in contrast, *0 bounce 00.080* (all pink bar) is a private unshared recipe.

middle, were used by only one player who chose not to share it with other players. Not surprisingly, the frequency of usage, indicated by the height of the bars, was significantly higher for the publicly shared recipes than the private ones.

Context Dependence. Observing the breadth of created recipes, it is clear that Foldit players use these algorithms to augment rather than to substitute for human strategizing. Players in essence perform a problem-informed search over the space of strategies, and use recipes to encode specific lower-level strategies. Different algorithms are employed at different stages in gameplay. Fig. 2 shows the relative frequency of recipe use in the opening, midgame, and endgame. Expert players exhibit different patterns of recipe use than the player population as a whole (compare Fig. 2 *A* and *B*). The top Foldit players use recipes *tlaloc Contract 3.00* and *Aotearoas_Romance* in the endgame, while most players use them almost equally at every stage in a puzzle (Fig. S2 *A* and *B*). Most Foldit players run *after-rebuild test* exclusively in the opening, but the top players often use it in the midgame as well. Local optimize recipes are heavily used in the endgame by top Foldit players (Fig. 2*B* and Fig. S2*F*).

Human strategic judgment plays an important role in choosing when and how to employ different recipes. Most recipes heavily rely on the interactive aspects of the game, and are used less as fully automated tools and more as power tools that serve as an extension of the player's strategy. We have not found any recipes that generate top models without human intervention; instead, Foldit players employ recipes to perform specific tasks at different stages of the folding process. For example, the local optimize recipes that walk along the protein backbone performing local minimizations are useless on the initial state of a Foldit puzzle that starts in an extended chain configuration, because a successful prediction will no longer have that backbone in an extended conformation and will require a new round of local optimization. Many of the aggressive rebuilding recipes require specifying which region of the protein to rebuild, as it is rarely useful to entirely rebuild a protein chain from beginning to end. These recipes are launched by players once they have converged to a low-energy solution and want to search nearby regions of conformational space before local optimization (Fig. S2 *C* and *D*); they are most useful in the midgame and often run for long periods without any human intervention. By contrast, all recipes in the set constraints category are designed to be run before launching other recipes or manually manipulating the protein (Fig. S2 *G* and *H*).

Overall, the context dependent use of different recipes is key to successful gameplay, but also makes it difficult to quantitatively evaluate the effectiveness of individual recipes because they are optimally used on quite different input protein states. This context dependence also complicates efforts to incorporate the rich new algorithms developed by Foldit players back into Rosetta or other automated methods.

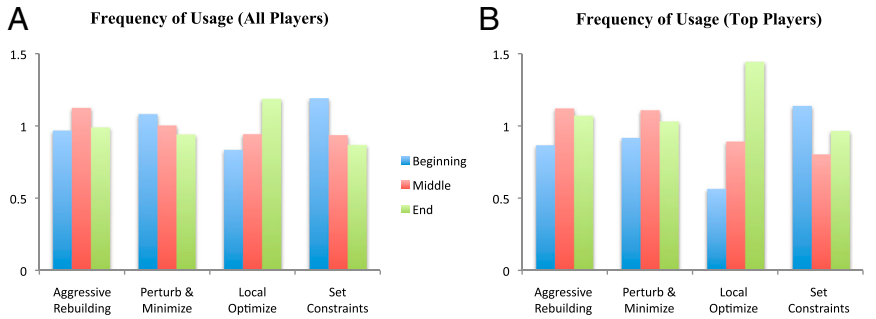


Fig. 2. Foldit players employ different recipes at different stages of gameplay. The relative frequency of usage for each recipe category is shown for three different stages of gameplay (blue, first third of gameplay; red, second third; green, final third) for all players (*A*) and for top ranked players (*B*). All players use local optimize recipes more frequently as gameplay progresses (*A*), and this trend is even stronger among top players (*B*). The usage frequency for each individual recipe is shown in detail in Fig. S2. The actual context dependence of the use of different recipes is likely greater than that shown in the figure; the division based on elapsed game time used here is relatively crude because different players in different puzzles may spend very different amounts of time on the opening, middle game, and end game.

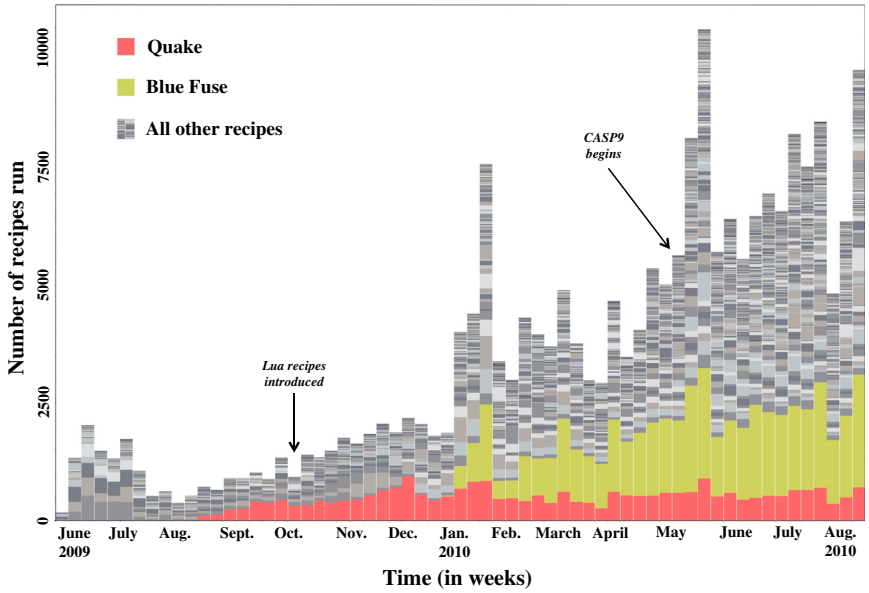


Fig. 3. Rapid proliferation of Foldit recipes between June 2009 and August 2010. Each different shade of gray represents an individual recipe with the size of the gray bar denoting how many times that recipe was run that particular week. The most heavily used script recipe is Blue Fuse, shown in yellow. The most popular GUI recipe is Quake, shown in red, which has been used consistently by Foldit players weekly since its creation.

Recipe Evolution. In addition to providing tools for recipe creation, we enabled sharing, evaluation, and refinement of recipes. These tools spurred a rapid evolution of recipe use in the Foldit player population from June 2009 [when the graphical user interface (GUI) recipes were first introduced; see *Materials and Methods*] through the end of August 2010. Fig. 3 shows the number of distinct recipes run every week across this time period, with different shades of gray representing different recipes. Two recipes were particularly heavily used: *Quake* (in red) and *Blue Fuse* (in yellow). The essential features of the algorithms contained within these two most popular recipes are similar. Quake is a GUI recipe that repeatedly packs the side chains and minimizes the side chain and backbone torsion angles, increasing and decreasing the strength of an applied set of rubber band constraints to promote annealing of the structure. The Blue Fuse algorithm (a script recipe; see *Materials and Methods*) is conceptually similar to but simpler than Quake: rather than varying the strength of artificial constraint functions, the strength of atom-atom repulsive interactions is alternately increased and decreased in repeated cycles of side chain packing and complete torsion angle minimization (Fig. S3). In both cases, an initially poorly packed structure is alternately compressed (by minimization with strong constraints or weak repulsive interactions) and then relaxed (by minimization with weak constraints or strong repulsive interactions); this alternation appears to be effective in locating low energy, well packed conformations.

Foldit algorithms evolve in a social fashion, with popular recipes copied and then elaborated upon. An evolutionary tree depicting the evolution of the Blue Fuse algorithm family is shown in Fig. 4; size corresponds to the logarithm of the number of recipe uses and color represents the recipe author. Popular recipes spawn large numbers of descendants, and there are multiple independent lineages each spanning many generations.

While the evolutionary mechanisms by which new algorithms are created (copying and variation) are clear, the mechanisms through which individual recipes spread through the population are less so. The popularity of Quake and Blue Fuse could result from individual player experimentation, player communication through chat, or player communication within teams. Interviews with Foldit players suggest that the primary mechanism by which successful algorithms take over the population is word of mouth. Players test and then recommend new recipes to others and rate recipes on the Foldit website, making decisions in part based on the reputation of the recipe author.

Similarity of Independently Developed Player and Scientist Algorithms. During this same time period, researchers in the Baker group were attempting to improve the core optimization methods within the Rosetta structure prediction and design program. A breakthrough was made with the *Fast Relax* algorithm, which achieves more effective energy optimization in less time than previous Rosetta methods. Benchmarks have shown Fast Relax to be considerably more efficient then an older algorithm, *Classic Relax* (12) used since 2002 (Fig. 5, compare green to red), and it is now used in all Rosetta de novo and homology modeling methods as well as enzyme design protocols.

The key innovation in the Fast Relax algorithm compared to Classic Relax and earlier methods is alternately increasing and decreasing the strength of the repulsive interactions. As diagrammed in Fig. 6, Fast Relax is comprised of interlaced cycles of full combinatorial repacking of the side chains of the protein (*repack*) and gradient-based minimization of the backbone and side chain degrees of freedom (*minimize*); within each round the repulsive weight is increased while from the end of one round to the start of the next (indicated by the arrow) the repulsive weight is decreased 50-fold. Typically 5–15 rounds of this repulsive

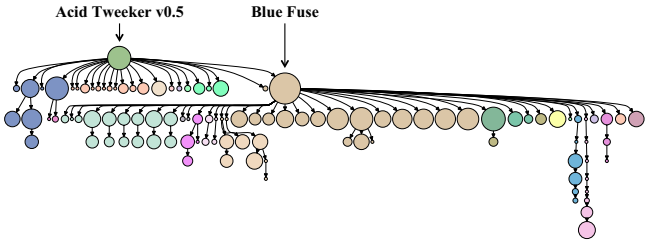


Fig. 4. Social evolution of Foldit recipes. Each circle represents a different Foldit recipe, each color denotes a different author, and the size is the logarithm of the number of times a recipe was used. Arrows represent the process of copying and subsequent modification: the recipe at the tip of an arrow was created from the recipe at the base of the arrow. *Acid Tweeker v0.5* is the parent of all of the recipes shown here with Blue Fuse its most popular offspring. The popularity of Blue Fuse led many Foldit players to create their own modified versions of the recipe.

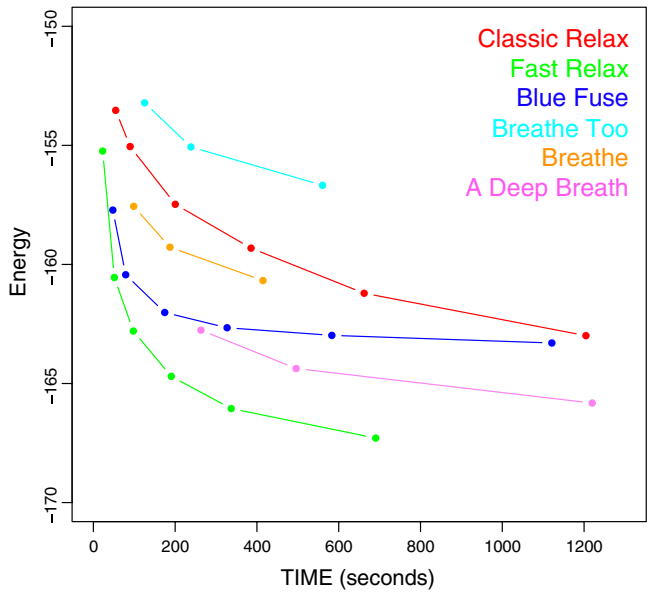


Fig. 5. Performance comparison between Rosetta Relax protocols and Foldit recipes. Each optimization protocol was run on a benchmark set of 6,758 models from 62 different proteins, and the average energy was calculated over all models after different numbers of iterations. The *x*-axis denotes the total CPU time in seconds and the *y*-axis represents the Rosetta energy. Foldit recipe Blue Fuse samples lower energies than Classic Relax and reaches these energies in much less time, but Fast Relax is even faster and more effective at energy optimization. A Deep Breath combines two other recipes (Breathe and Breathe Too) with Blue Fuse, resulting in a longer average runtime but lower overall energies compared to Blue Fuse.

weight annealing are applied to a given structure and the lowest energy structure encountered (only full repulsive weight structures are eligible) is finally kept as the result.

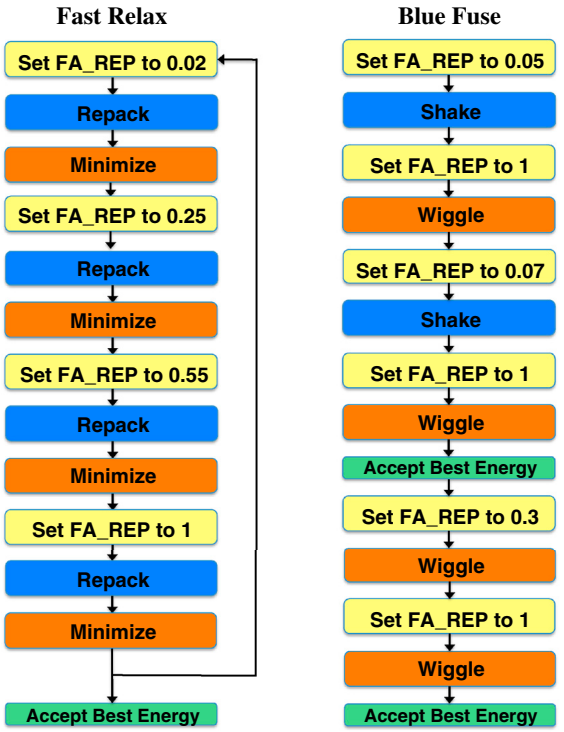


Fig. 6. Similarity between the unpublished Rosetta protocol Fast Relax and the Foldit recipe Blue Fuse. Equivalent steps in the two algorithms are represented with the same color. The repulsive interaction strength is reset from high to low in each cycle of Fast Relax; in Rosetta 5–15 iterations are typically used.

There is a striking similarity between the Fast Relax algorithm developed by scientists and the Blue Fuse algorithm developed by Foldit game players. These similarities are evident in the algorithm comparison shown in Fig. 6. Both algorithms ramp the repulsive weight up and down while repacking and minimizing the structure, ultimately selecting as the algorithm output the lowest energy structure encountered. There are minor differences—Blue Fuse begins with a low repulsive weight and only performs a shake before minimizing the structure at the standard weight while Fast Relax does a repack/minimize cycle at each stage—but the similarities far outweigh the differences.

The Foldit players' algorithmic discovery is in a rich tradition of softening the repulsive forces to enhance sampling in protein folding calculations. The earliest methods replaced entire subsets of atoms with simplified centroid side chains (11), and subsequent methods softened repulsive interactions in full atom representations (13). Classic Relax expanded on this approach by gradually ramping up the repulsive term during the course of a simulation. Through social evolution of recipes, Foldit players independently rediscovered the utility of initially softening the repulsive interactions. The players went beyond previous approaches by introducing the sawtooth-like repeated annealing of the strength of the repulsive interactions as in Fast Relax. This sawtooth profile likely induces repeated compaction and expansion of the protein chain, which evidently helps access new energy minima.

Performance Comparison. To determine how the Blue Fuse algorithm compared to both the Classic Relax and Fast Relax protocols, we ran all three algorithms on an in-house standard benchmark set consisting of 62 proteins with a range of structural diversity, including monomeric as well as multimeric structures with and without ligands. For each protein we included both native and close-to-native structures as well as nonnative Rosetta models for a total of 6,758 structures. The CPU time required for each of the methods can be varied by changing the number of iterations in the outer loop, and we recorded the average energy over all of the 6,758 models as a function of CPU time. Fig. 5 shows that the Blue Fuse algorithm (in blue) performs more efficiently than Classic Relax (in red), but not as well as the newer Fast Relax protocol (in green). Thus, the algorithm encoded in the most popular player recipe is not only structurally similar to the Fast Relax but also more efficient than previously published Rosetta algorithms.

Rosetta optimizations are simplified to run at interactive speeds suitable for human exploration in Foldit (*SI Text*), and hence Fast Relax in Rosetta utilizes more powerful elementary optimization modules than Blue Fuse in Foldit. To evaluate the performance of Fast Relax relative to Blue Fuse–when given access only to the simplified optimizations in Foldit that are available to game players–we encoded the Fast Relax algorithm using the Foldit script interface. Fig. 7 shows the performance of this Foldit script version of Fast Relax (in dark green) on the same benchmark set; although it is still able to sample lower energies than Blue Fuse, it takes longer to do so. Notably, Blue Fuse outperforms this Foldit version of Fast Relax for CPU times less than 200 s (Fig. S4), and in practice Foldit players run Blue Fuse for an average runtime of 122 s (dotted line in Fig. 7, Fig. S4). Thus, given the tools available in the game, and for the typical times these tools are used within the game, the player discovered Blue Fuse algorithm is actually superior to Fast Relax.

We tested other popular recipes (shown in Table S1), but found none that minimized the energy as quickly as Blue Fuse. Many recipes yielded lower energies than Blue Fuse but took much longer to do so. While repeated iterations of Blue Fuse fail to decrease the energy further, Foldit players discovered that by combining different recipes together with Blue Fuse, lower energies can be achieved than with any of them alone; for example, *A Deep Breath* in Fig. 5. Again, context dependence is important:

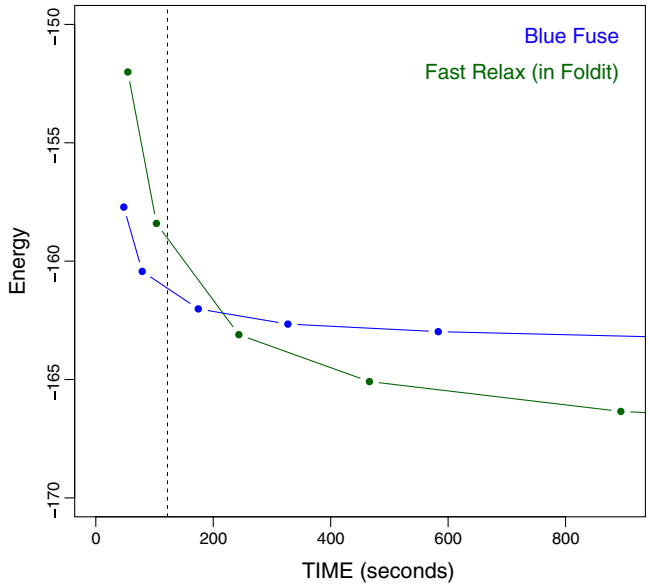


Fig. 7. Performance comparison between Blue Fuse and Rosetta Fast Relax protocol using the streamlined Foldit minimization and side chain optimization routines. The dark green line shows the performance of Fast Relax encoded using the Foldit scripting interface. Fast Relax still samples lower energies than Blue Fuse, but takes longer to do so than Fast Relax in Rosetta. Blue Fuse is more effective than the Foldit version of Fast Relax on time scales most compatible with gameplay: the average Blue Fuse runtime during actual gameplay of 122 s is indicated by the dotted line.

the authors of A Deep Breath, for example, recommend adding rubber bands to a protein before running.

Discussion

The introduction of tools for Foldit players to codify their strategies spawned a flurry of creative activity and the creation of a remarkable diversity of folding algorithms. This diversity emerged through the evolution of player recipes over the course of a year since their introduction. Particularly remarkable is the convergence of this evolution onto an algorithm similar to one recently developed by scientific experts over the same period of time. Both algorithms achieve faster and more effective energy optimization than previous methods, and within the context of the game, the player algorithm is the most efficient. Foldit players have been at a considerable disadvantage compared to scientists in developing new algorithms as the scripting language only exposes a small fraction of the variables and base algorithms in the Rosetta codebase. We are now generalizing the scripting language to allow control over more of these features, and look forward to learning what Foldit player ingenuity can do with these additional capabilities. More generally, the explosion of Foldit algorithms and the convergence on the best algorithm discovered thus far by scientists highlights the potential of scientific discovery games for significant contributions to science and technology, particularly in the creation and formalization of complex problem-solving strategies.

Materials and Methods

In order to empower the widest possible pool of active players to create recipes, including those without basic programming skills, we provided two recipe creation pathways. The first recipe creation tool, provided to Foldit players in June 2009, was a simple block-based visual programming interface where different actions are added from a pull-down menu. Available actions include adding or adjusting the strength of rubber bands (soft constraints which connect amino acids and pull on them independently of the player), restoring previous structures (allowing backtracking), and invoking optimization methods such as shake and wiggle. Recipes created using this initial GUI (referred to as *GUI recipes*) could not utilize conditionals or loops, or modify certain properties such as the *clash importance*, which lets players adjust the strength of the Rosetta atom-atom steric overlap energy term. To support more advanced recipes, we added a text-based interface, using the Lua scripting language (14), to the game in October 2009. In addition to having many more Foldit actions available, this interface gives players the ability to control the flow of recipes (using conditionals and loops), allowing for the creation of much more complex algorithms (these are referred to as *script recipes*).

We analyzed the evolution and use of Foldit recipes during the CASP9 structure prediction experiment from May–August 2010 (15). Because Foldit is an online game, we were able to track recipe usage and analyze the recipes developed and employed by Foldit players during this period.

Availability

Foldit player recipes are available on the Foldit website: http://fold.it/portal/recipes, with instructions on how to download them described on the Foldit wiki: http://foldit.wikia.com/wiki/101_-_Cookbook.

ACKNOWLEDGMENTS. We thank the members of the Foldit team for their help designing and developing the game and all the Foldit players who have made this work possible. This work was supported by the Center for Game Science, Defense Advanced Research Projects Agency (DARPA) Grant N00173-08-1-G025, the DARPA Protein Design Processes (PDP) program, National Science Foundation (NSF) Grants IIS0811902 and IIS0812590, the Howard Hughes Medical Institute (D.B.), a Henry Wellcome Postdoctoral Fellowship (M.D.T), Adobe and Microsoft. This material is based upon work supported by the National Science Foundation under Grant 0906026.

1. Lintott C, et al. (2009) Galaxy Zoo: 'Hanny's Voorwerp', a quasar light echo? *Mon Not R Astron Soc* 399:129–140.
2. Westphal AJ, et al. (2010) Non-destructive search for interstellar dust using synchrotron microprobes. *X-ray Optics and Microanalysis: Proceedings of the 20th International Congress*, CP1221 (American Institute of Physics (AIP), Melville, NY), pp 131–138.
3. *NASA: Be a Martian*, http://beamartian.jpl.nasa.gov/welcome.
4. Cooper S, et al. (2010) The challenge of designing scientific discovery games. *Proceedings of the Fifth international Conference on the Foundations f Digital Games*, 10 (FDG ACM, New York, NY), pp 40–47.
5. Cooper S, et al. (2010) Predicting protein structures with a multiplayer online game. *Nature* 466:756–760.
6. Rohl CA, Strauss CE, Misura KM, Baker D (2004) Protein structure prediction using Rosetta. *Methods Enzymol* 383:66–93.
7. Khatib F, et al. (2011) Crystal structure of monomeric retroviral protease solved by protein folding game players. *Nat Struct Mol Biol* 18:1175–1177.
8. *Foldit Wiki*, http://Foldit.wikia.com/.
9. Banerji M, et al. (2010) Galaxy Zoo: Reproducing galaxy morphologies via machine learning. *Mon Not R Astron Soc* 406:342–353.
10. Cooper S, et al. (2011) Analysis of social gameplay macros in the Foldit cookbook. *Proceedings of the Sixth international Conference on the Foundations f Digital Games*, 11 (FDG ACM, New York, NY).
11. Levitt M, Warshel A (1975) Computer simulation of protein folding. *Nature* 253:694–698.
12. Kuhlman B, et al. (2003) Design of a novel globular protein fold with atomic-level accuracy. *Science* 302:1364–1368.
13. Levitt M (1983) Protein folding by restrained energy minimization and molecular dynamics. *J Mol Biol* 170:723–764.
14. *The Programming Language Lua*, http://www.lua.org/.
15. *CASP*, http://predictioncenter.org/.

Supporting Information

Khatib et al. 10.1073/pnas.1115898108

SI Text

Simplified Foldit Tools. The Rosetta optimizations *minimize* and *repack* are simplified in Foldit to run at interactive speeds because as the number of residues in a protein increases, running optimizations over all of them simultaneously becomes too slow.

Instead of combinatorial optimization of all side chain rotamer conformations simultaneously carried out by Rosetta repack, for example, *shake* randomly selects a residue, repacks a neighborhood of surrounding residues, then selects another random residue and repeats the process until every rotamer has been repacked. Running these shorter optimizations in sequence rather than one long Rosetta repack allows Foldit players to receive immediate partial feedback with the ability to cancel the optimization at any time.

Wiggle differs from Rosetta minimize (quasi-Newton optimization of all backbone and side chain torsion angles) in that it optimizes different subsets of the protein's degrees of freedom on different iterations. On some iterations, wiggle allows interresidue degrees of freedom to vary, but heavily constrains them to maintain ideality. Wiggle also automatically updates the protein's internal atomic neighbor list as the protein may move significantly during a wiggle.

Quake & Blue Fuse. Foldit players have consistently used both these recipes every week since their creation; Quake is the most popular graphical user interface (GUI) recipe while Blue Fuse is the most popular script recipe. Blue Fuse was run over 24,000 times by Foldit players, more than three times as much as the next highest used recipe (Quake). Although certain players ran Blue Fuse much more than others (the creator of Blue Fuse used it over 2,000 times), this recipe was the clear favorite with 105 Foldit players running it at least 10 times. The GUI recipe Quake was the most heavily used recipe before script recipes were introduced into Foldit. At times Quake accounted for over 40% of all recipe usage in Foldit (Fig. S5), a feat which no other recipe has since matched.

GUI recipes did not initially have the functionality to change the clash importance, so players had to come up with alternative methods to overcome repulsive forces and perturb the protein. Foldit player *Grom*, who wrote Quake in August 2009, noticed that it was often possible to get better energies by pushing the protein in some way and then doing a shake/wiggle or wiggle/shake combination. The easiest way to implement this move in an automated GUI recipe was to use rubber bands.

Quake uses only three different Foldit tools: shake, wiggle, and rubber bands. A screenshot of the first part of the GUI recipe Quake is shown in Fig. S6. Rubber bands are applied every 30 residues, starting at the 1st and 15th residues, and the entire protein is minimized using these soft constraints, compacting the structure. The rubber bands are then removed and a series of shakes and wiggles are performed to find the closest local energy minimum. This process is repeated again, starting with the 6th and 21st residues, before moving onto a similar procedure involving fewer rubber bands but increasing the strength of them. A flowchart describing the Quake algorithm is presented in Fig. S7.

Once script recipes were added to Foldit, many players took advantage of the fact that they were able to adjust the strength of the repulsion in their recipes. Acid Tweeker v0.5, written by *Steven Pletsch* in December 2009, was one of those script recipes and was used as a template by *vertex*, the author of Blue Fuse. When a Foldit player publicly shares a recipe, any other player can modify their recipe; this new recipe becomes a child of the original parent recipe. Fig. 4 shows all the recipe descendants from Acid Tweeker v0.5 and Blue Fuse where size is the logarithm of the number of recipe uses and each color represents the different Foldit players who wrote each recipe. Many different players used Acid Tweeker v0.5 as a parent, but none of its children have been as popular as Blue Fuse, which in turn has had many descendants. The Blue Fuse algorithm is similar but simpler than Quake, with the clash importance rather than bands annealed.

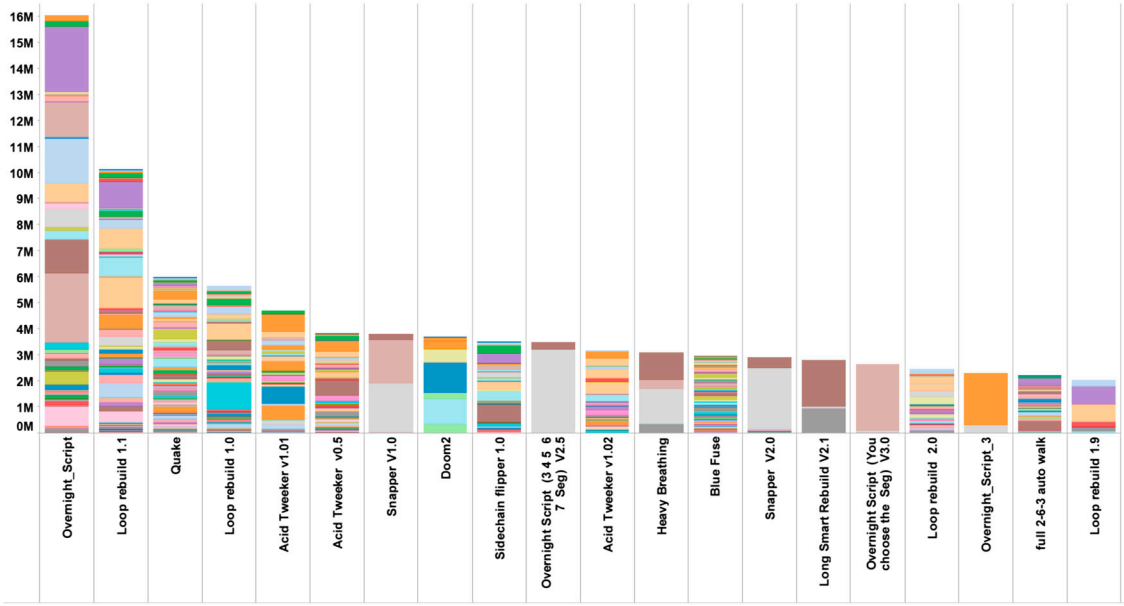
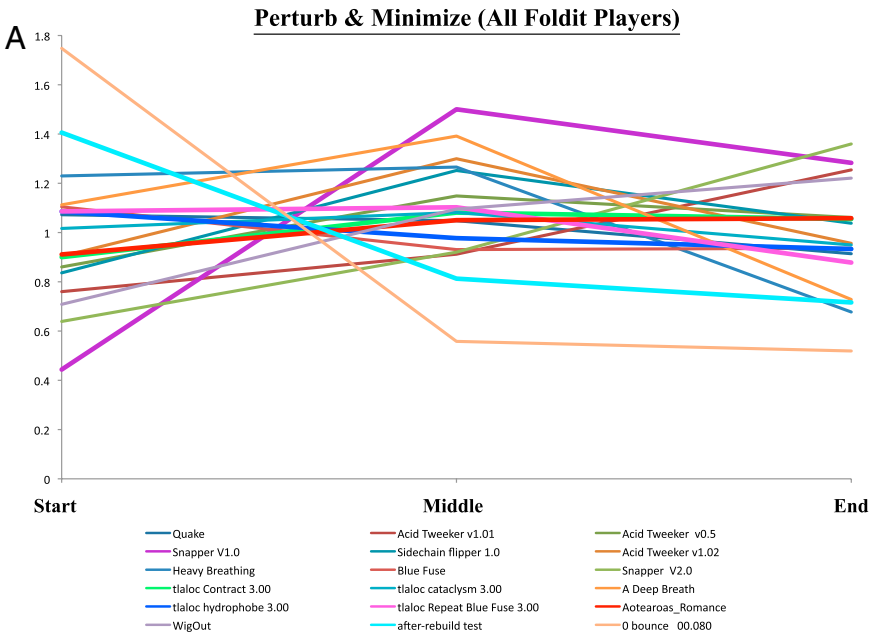
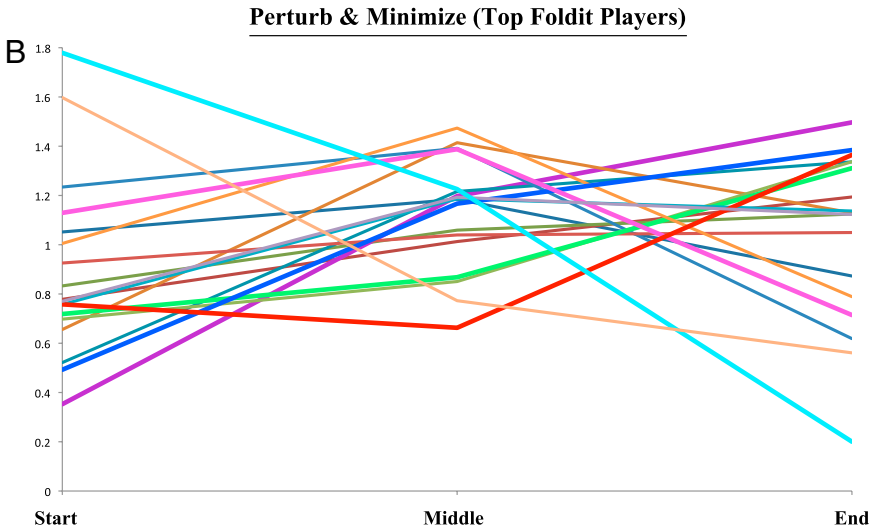
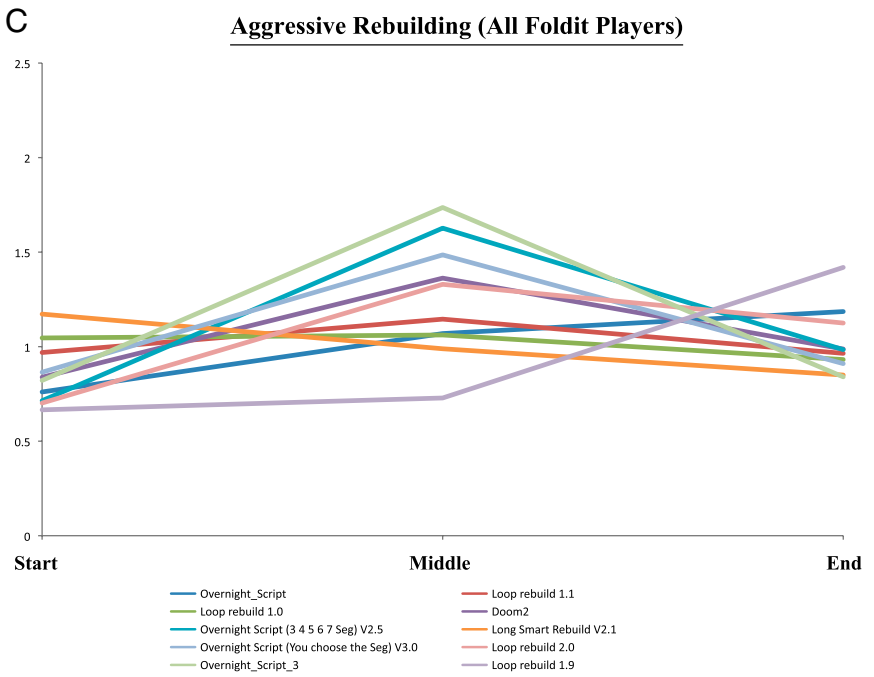


Fig. S1. Usage statistics (by CPU time) of Foldit recipes run. Each bar on the *x*-axis represents a different Foldit recipe with the height of the bar denoting the amount of CPU time (in seconds) each recipe was used. Each color represents a different Foldit player who used that recipe. At the very left, Overnight_Script was run for over 16 million CPU seconds.

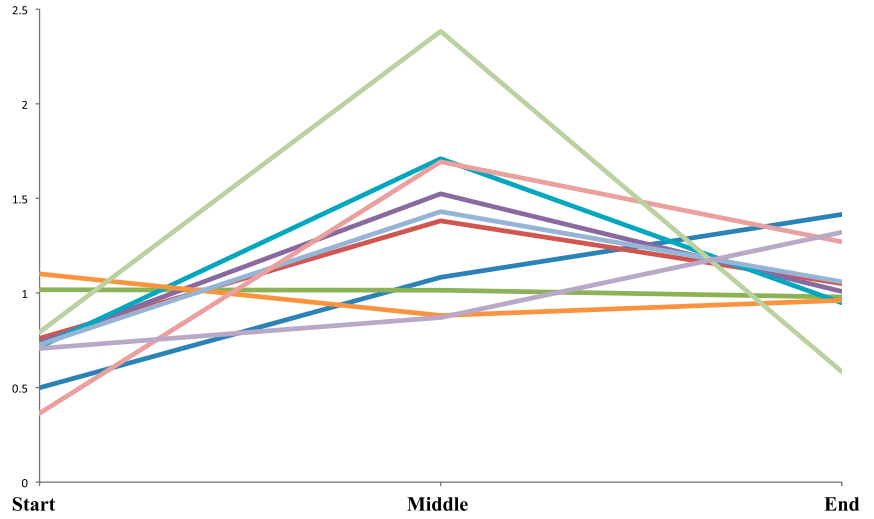

A
Perturb & Minimize (All Foldit Players)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Start
Middle
End
Quake
Acid Tweeker v1.01
Acid Tweeker v0.5
Snapper V1.0
Sidechain flipper 1.0
Acid Tweeker v1.02
Heavy Breathing
Blue Fuse
Snapper V2.0
tlaloc Contract 3.00
tlaloc cataclysm 3.00
A Deep Breath
tlaloc hydrophobe 3.00
tlaloc Repeat Blue Fuse 3.00
Aotearoas_Romance
WigOut
after-rebuild test
0 bounce 00.080


B
Perturb & Minimize (Top Foldit Players)
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Start
Middle
End

C


Aggressive Rebuilding (All Foldit Players)
2.5
2
1.5
1
0.5
0
Start
Middle
End
Overnight_Script
Loop rebuild 1.0
Overnight Script (3 4 5 6 7 Seg) V2.5
Overnight Script (You choose the Seg) V3.0
Overnight_Script_3
Loop rebuild 1.1
Doom2
Long Smart Rebuild V2.1
Loop rebuild 2.0
Loop rebuild 1.9

D


Aggressive Rebuilding (Top Foldit Players)
2.5
2
1.5
1
0.5
0
Start
Middle
End

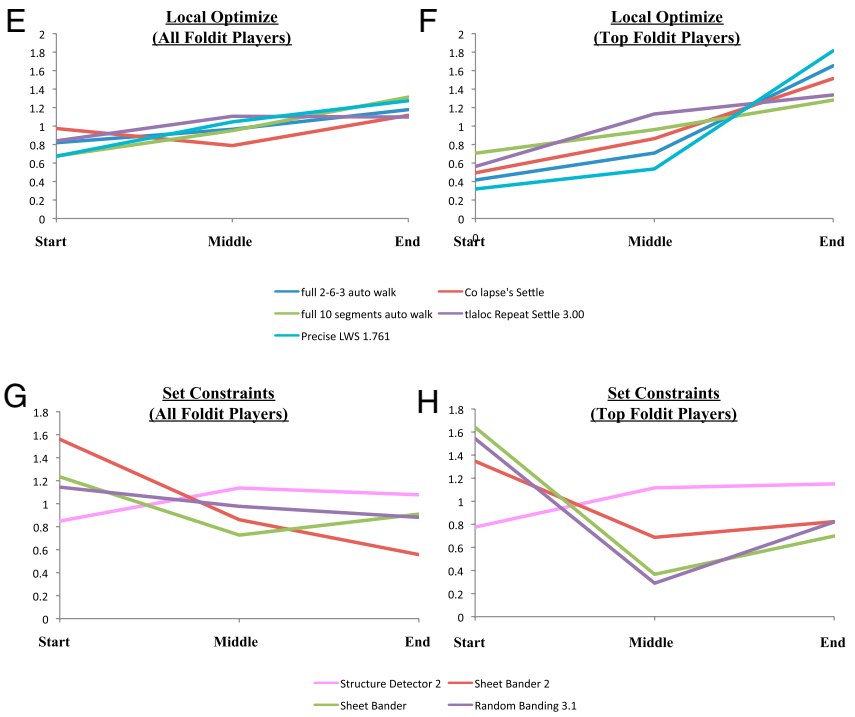


Fig. S2. Foldit players employ different recipes at different stages of gameplay. Each colored line represents a different Foldit recipe, and the *y*-axis indicates the frequency of usage of the recipe at different stages of gameplay. (*A*) Recipe usage frequencies for perturb & minimize recipes for all Foldit players and (*B*) for top ranked players. Most players use many of these recipes equally at every stage in a puzzle, whereas top ranked players use some of them mostly in the end game (such as tlaloc Contract 3.00 and Aotearoas_Romance, shown in green and red), mostly in the middle of a puzzle (tlaloc Repeat Blue Fuse 3.00, shown in pink) or either in the middle or end game (tlaloc hydrophobe 3.00, shown in blue). Most Foldit players run after-rebuild test exclusively in the opening, but the top players often use it in the midgame as well (shown in cyan). Snapper V1.0 is most-often run in the middle of a puzzle by most Foldit players, but used highly in the end game by top ranked players (shown in purple). (*C*) Recipe usage frequencies for aggressive rebuilding recipes for all Foldit players and (*D*) for top ranked players. Most rebuild recipes are used in the middle of a puzzle. (*E*) Recipe usage frequencies for local optimize recipes for all Foldit players and (*F*) for top ranked players. These recipes are generally used in the endgame by the top players. (*G*) Recipe usage frequencies for set constraints recipes for all Foldit players and (*H*) for top ranked players. Scripts involving the imposition of specific sets of constraints were used primarily in the opening, whereas the *Structure Detector 2* recipe (shown in pink), which assigns secondary structure to a protein chain that has none provided, was most often used in the middle and end-game.

```
reset_recent_best()
select_all()
set_behavior_clash_importance(.05)
do_shake(1)
set_behavior_clash_importance(1)
do_global_wiggle_all(8)
set_behavior_clash_importance(.07)
do_shake(1)
set_behavior_clash_importance(1)
do_global_wiggle_all(8)
restore_recent_best()
set_behavior_clash_importance(.3)
do_global_wiggle_all(1)
set_behavior_clash_importance(1)
do_global_wiggle_all(8)
restore_recent_best()
```

Fig. S3. Example of a Foldit script recipe. Screenshot of the entire Blue Fuse algorithm.

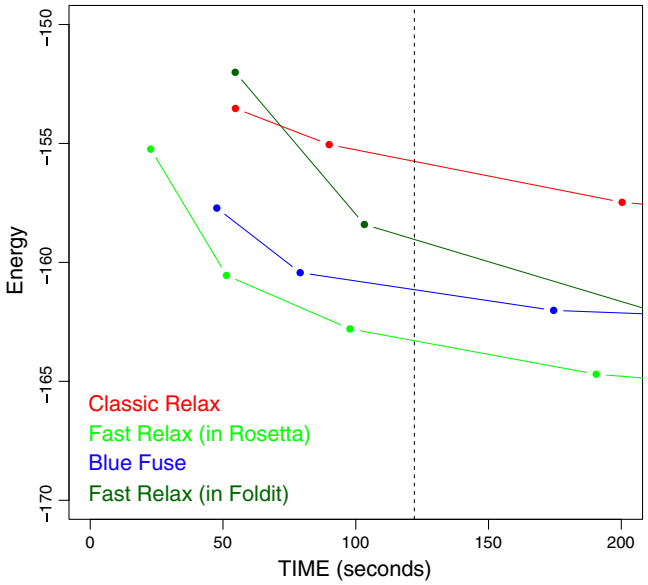


Fig. S4. Performance comparison, under 200 s, between Blue Fuse and Rosetta Fast Relax protocol rewritten in Foldit's Lua script interface. Foldit players ran Blue Fuse with an average runtime of 122 s (shown by the dotted line). The Foldit recipe Blue Fuse has better energy optimization, within the first 200 s on average, compared to the version of Fast Relax written using the equivalent Foldit tools.

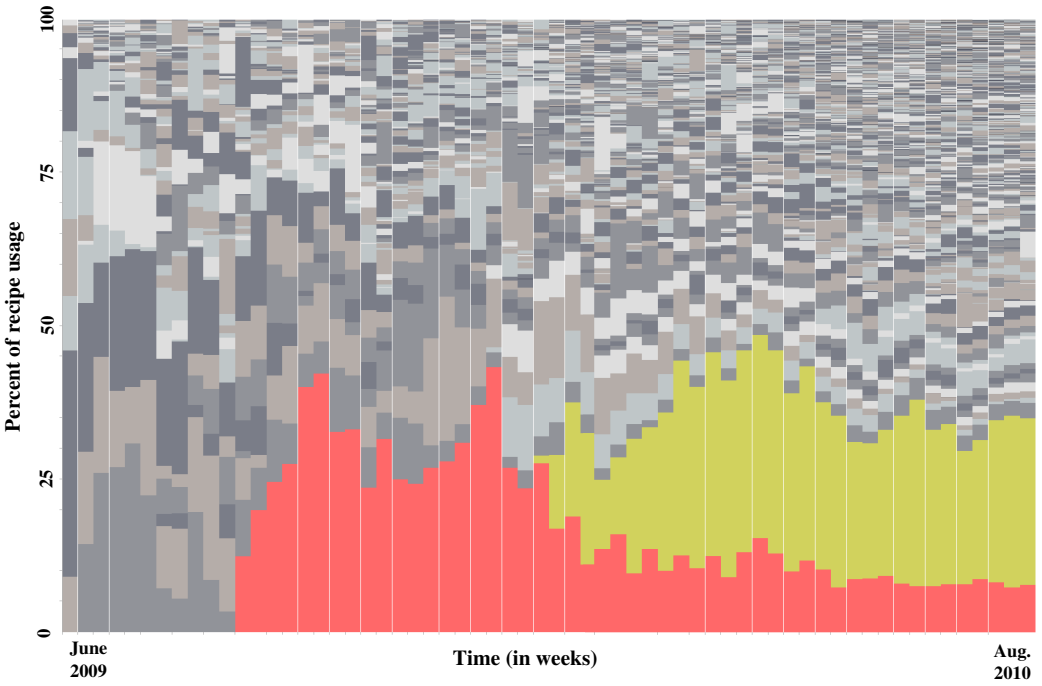


Fig. S5. Percentage of total Foldit recipe usage per week from June 2009 through the end of August 2010. Each different shade of gray represents an individual recipe with the size of the gray bar denoting how often a recipe was run that particular week as a percentage of total recipe usage. Before the creation of Blue Fuse, shown in yellow, Quake, in red, at times accounted for over 40% of recipe usage in Foldit.

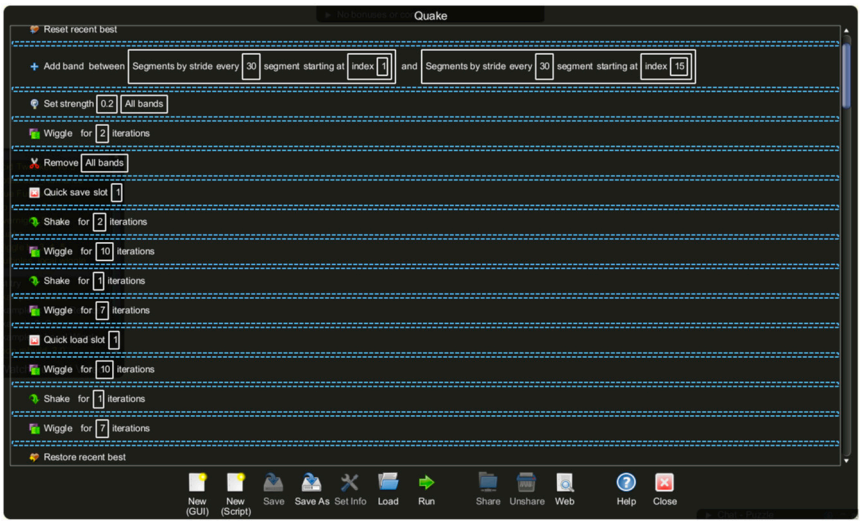


Fig. S6. Example of a Foldit GUI recipe. Screenshot of the first round of the Quake algorithm shown in the Foldit GUI recipe editor.

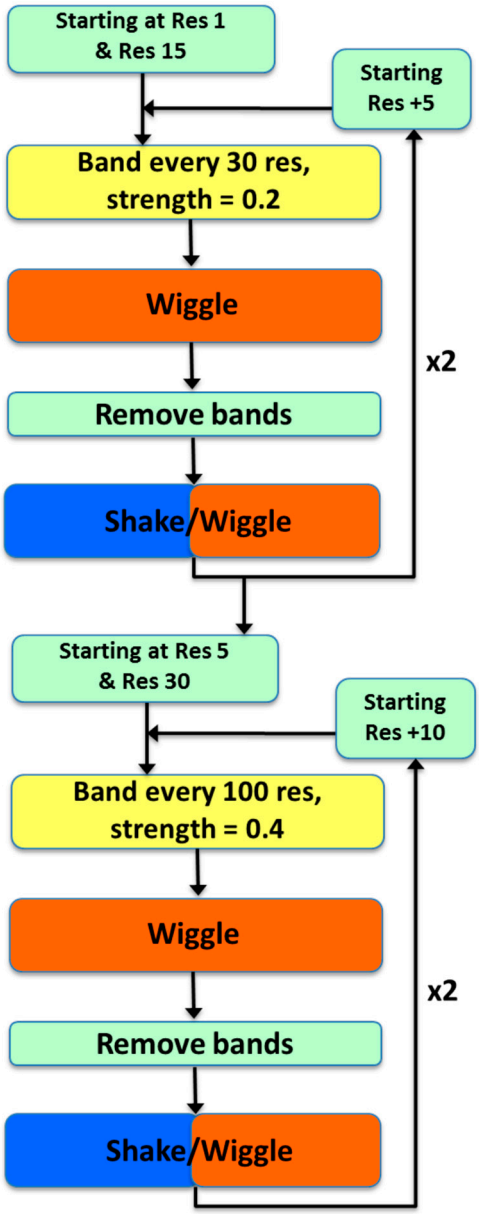


Fig. S7. Quake recipe. Flowchart of the Quake algorithm.

Table S1. Statistics on the most used Foldit recipes

Recipe name	CPU (days)	Usage	Users	Type	Category	Author
Overnight_Script	185.49	1,324	72	script	aggressive rebuilding	Steven Pletsch
Loop rebuild 1.1	117.16	2,233	110	script	aggressive rebuilding	spvincent
Quake	69.00	7,772	181	GUI	perturb & minimize	Grom
Loop rebuild 1.0	65.14	2,082	85	script	aggressive rebuilding	spvincent
Acid Tweeker v1.01	54.30	636	50	script	perturb & minimize	Pletsch
Acid Tweeker v0.5	44.29	1,508	92	script	perturb & minimize	Steven Pletsch
Snapper V1.0	44.07	193	5	script	perturb & minimize	TheGUmmer
Doom2	42.68	492	13	script	aggressive rebuilding	Ocire
Side chain flipper 1.0	40.40	1,058	61	script	perturb & minimize	spvincent
Overnight Script (3 4 5 6 7 Seg) V2.5	38.02	178	2	script	aggressive rebuilding	TheGUmmer
Acid Tweeker v1.02	36.42	557	51	script	perturb & minimize	Pletsch
Heavy Breathing	35.65	538	5	script	perturb & minimize	TheGUmmer
Blue Fuse v1.1 *	34.28	24,273	180	script	perturb & minimize	vertex
Snapper V2.0	33.48	268	6	script	perturb & minimize	TheGUmmer
Long Smart Rebuild V2.1	32.25	2,012	3	script	aggressive rebuilding	TheGUmmer
Overnight Script (You choose the Seg) V3.0	30.56	213	3	script	aggressive rebuilding	TheGUmmer
Loop rebuild 2.0	28.45	489	46	script	aggressive rebuilding	spvincent
Overnight_Script_3	26.49	139	2	script	aggressive rebuilding	Pletsch
Full 2-6-3 auto walk	25.28	1,230	57	GUI	local optimize	Madde
Loop rebuild 1.9	23.24	265	8	script	aggressive rebuilding	spvincent
tlaloc Contract 3.00	20.42	2,257	112	script	perturb & minimize	Tlaloc
tlaloc Cataclysm 3.00	17.58	1,294	106	script	perturb & minimize	Tlaloc
A Deep Breath	10.55	1,418	8	script	perturb & minimize	TheGUmmer
Co lapse's Settle	10.20	2,380	49	GUI	local optimize	Co lapse
full 10 segments auto walk	6.93	1,083	35	GUI	local optimize	Madde
tlaloc Repeat Settle 3.00	6.06	1,096	89	script	local optimize	Tlaloc
tlaloc Hydrophobe 3.00	5.77	1,970	110	script	perturb & minimize	Tlaloc
tlaloc Repeat Blue Fuse 3.00	5.29	1,785	112	script	perturb & minimize	Tlaloc
Aotearoas_Romance	4.74	1,011	92	GUI	perturb & minimize	Aotearoa
WigOut	3.93	1,322	7	script	perturb & minimize	CharlieFortsConscience
Precise LWS 1.761	3.77	1,067	55	script	local optimize	Rav3n_pl
after-rebuild test	1.97	1,458	6	GUI	perturb & minimize	LennStar
Structure Detector 2	0.75	1,261	69	script	set constraints	Enzymatic2_0
0 bounce 00.080	0.16	1,281	1	script	perturb & minimize	vertex
Sheet Bander 2	0.08	1,094	42	script	set constraints	Enzyme
Sheet Bander	0.04	1,160	25	script	set constraints	Enzymatic
Random Banding 3.1	0.02	1,102	40	script	set constraints	Crashguard303

The first 20 recipes represent those that were used for the longest time (all recipes run for over 2 million CPU seconds), indicated by the total number of CPU days in the second column. The third column denotes the number of times each recipe was launched by Foldit players; all recipes highlighted in Fig. 1, those that were run over 1,000 times, are included in this table. The fourth column represents the distinct number of Foldit players who used each recipe. The fifth column specifies whether a recipe was created using the GUI interface or the more powerful Lua script interface. The sixth column breaks down each recipe into four different categories: perturb and minimize, aggressive rebuilding, local optimize, and set constraints. The final column indicates the author of the recipe by Foldit username

*Note: We refer to the Foldit recipe *Blue Fuse v1.1* as Blue Fuse throughout the text for simplicity.