Evolving Solvers for FreeCell and the Sliding-Tile Puzzle

Achiya Elyasaf
Ben-Gurion University of the Negev, Be'er Sheva, Israel
achiya.e@gmail.com

Yael Zaritsky *
Ben-Gurion University of the Negev, Be'er Sheva, Israel
yaelzaritsky@gmail.com

Ami Hauptman
Ben-Gurion University of the Negev, Be'er Sheva, Israel
amihau@gmail.com

Moshe Sipper
Ben-Gurion University of the Negev, Be'er Sheva, Israel
sipper@cs.bgu.ac.il

Abstract

We use genetic algorithms to evolve highly successful solvers for two puzzles: FreeCell and Sliding-Tile Puzzle.

Discrete puzzles, also known as single-player games, are an excellent problem domain for artificial intelligence research, because they can be parsimoniously described yet are often hard to solve (Pearl 1984). A well-known, highly popular example within the domain of discrete puzzles is the card game of FreeCell. Another highly popular game is the sliding-tile puzzle, the traditional versions of which are the 15-puzzle (4X4) and the 24-puzzle (5X5). State-of-the-art heuristics allow for fast solutions of arbitrary instances of the 15-puzzle (4X4), however the 24-puzzle (5X5) remains computationally challenging.

FreeCell remained relatively obscure until it was included in the Windows 95 operating system (and in all subsequent versions), along with 32,000 problems—known as *Microsoft 32K*—all solvable but one (this latter, game #11982, was proven to be unsolvable). Despite there being numerous FreeCell solvers available via the Web, few have been written up in the scientific literature. The best published solver to date is that of Heineman (Heineman 2009), able to solve 96% of Microsoft 32K using a hybrid A* / hill-climbing search algorithm called *staged deepening* (henceforth referred to as the *HSD* algorithm). The HSD algorithm, along with a heuristic function, forms Heineman's FreeCell solver (we shall distinguish between the HSD algorithm, the HSD heuristic, and the HSD solver—which includes both). Heineman's system exploits several important characteristics of the game, elaborated below.

Search algorithms for puzzles (as well as for other types of problems) are strongly based on the notion of approximating the distance of a given configuration (or *state*) to the problem's solution (or *goal*). Such approximations are found by means of a computationally efficient function, known as the *heuristic function*.

In a recent work Hauptman et al. (Hauptman et al. 2009) successfully applied genetic programming (GP) to *evolving* heuristic functions for the Rush Hour puzzle—a hard, PSPACE-Complete puzzle. The evolved heuristics dramatically reduced the amount of nodes traversed by an enhanced "brute-force," iterative-deepening search algorithm. Herein, we employ a genetic algorithm (GA) to obtaining solvers for both the difficult FreeCell puzzle and the sliding-tile puzzle. Note that although from a computational-complexity point of view the Rush Hour puzzle is harder (unless *NP=PSPACE*), search spaces induced by *typical* instances of FreeCell tend to be substantially larger than those of Rush Hour, and thus far more difficult to solve. This is evidenced by the failure of standard search methods to solve FreeCell, as opposed to their success in solving all 6x6 Rush Hour problems without requiring any heuristics (Hauptman et al. 2009).

Our main set of experiments focused on evolving combinations of hand-crafted heuristics we devised specifically for FreeCell (Elyasaf, Hauptman, and Sipper 2011). We used these basic heuristics as building blocks in a GA setting, where individuals represented the heuristics' weights. We used Hillis-style competitive coevolution (Hillis 1990) to simultaneously coevolve good solvers and various deals of varying difficulty levels.

We will show that not only do we solve 98% of the Microsoft 32K problem set, a result far better than the best solver on record, but we also do so significantly more efficiently in terms of time to solve, space (number of nodes expanded), and solution length (number of nodes along the path to the correct solution found).

Since Heinman's algorithm outperformed both A* and IDA*, we turned to the HSD algorithm. The algorithm is described in (Heineman 2009). We next describe some of the heuristics we used. Note that not all of them are heuristics in the strict sense, in that they do not compute an estimated distance to the goal.

Heineman's Staged Deepening Heusirtic (HSDH). For each foundation pile locate within the cascade piles the next card that should be placed there, and count the cards found on top of it. The returned value is the sum of this count for all foundations. This number is multiplied by 2 if there are no free FreeCells or empty foundation piles (reflecting the fact that freeing the next card is harder in this case).

NumberWellPlaced. Count the number of *well-placed* cards in cascade piles. A pile of cards is well placed if *all* its cards are in descending order and alternating colors.

NumCardsNotAtFoundations. Count the number of cards

*Yael Zaritsky is partially supported by the Lynn and William Frankel Center for Computer Sciences.

that are not at the foundation piles.

Examples of some of the heuristics for the sliding-tile puzzle are:

ManhattanDistance. For each tile count how many moves it will take to get to its goal position, assuming it is the only tile on the board.

GeometricMean. The geometric mean of the Manhattan distances of all tiles.

LinearConflict. Two tiles t_j and t_k are in a linear conflict if t_j and t_k are situated along the same horizontal line, the goal positions of t_j and t_k are both along that line, t_j is to the right of t_k, and the goal position of t_j is to the left of the goal position of t_k. Every linear conflict increments the heuristic value by one.

PDB. All of the 4-tile combinations of PDBs for the 15-puzzle were taken as different heuristics.

Experiments with each of these heuristics demonstrated that most of them were not good enough to guide search for these difficult domains. Thus we turned to evolution.

Combining several heuristics to get a more accurate one is considered one of the most difficult problems in contemporary heuristics research (Burke et al. 2010; Samadi, Felner, and Schaeffer 2008). Herein we tackle a sub-problem, that of combining heuristics by *arithmetic* means, e.g., by summing their values or taking the maximal value.

Each individual comprises n real values in the range $[0, 1)$, representing a linear combination of all n heuristics. Specifically, the heuristic value, H, designated by an evolving individual is defined as $H = \sum_{i=1}^{n} w_i h_i$, where w_i is the ith weight specified by the genome, and h_i is the ith heuristic. To obtain a more uniform calculation we normalized all heuristic values to within the range $[0, 1]$ by maintaining a maximal possible value for each heuristic, and dividing by it.

We used standard fitness-proportionate selection and single-point crossover. Mutation was performed in a manner analogous to bitwise mutation by replacing with independent probability 0.1 a (real-valued) weight by a new random value in the range $[0, 1)$.

A FreeCell individual's fitness score was obtained by performing full HSD search on deals taken from the training set, with the individual used as the heuristic function. Fitness equaled the average search-node reduction ratio. This ratio was obtained by comparing the reduction in number of search nodes—averaged over solved deals—with the average number of nodes when searching with the original HSD heuristic (HSDH).

We used Hillis-style of coevolution wherein a population of solutions coevolves alongside a population of problems (Hillis 1990). The basic idea is that neither population should stagnate: As solvers become more adept at solving certain problems these latter do not remain in the problem set but are rather removed from the population of problems—which itself evolves. In this form of competitive coevolution the fitness of one population is inversely related to the fitness of the other population.

In our coevolutionary scenario the first population comprises the solvers, as described above. In the second population an individual represents a *set* of 6 FreeCell deals selected from Microsoft 32K. The genome and genetic operators of the both populations were identical to those defined above.

Our top evolved individual, dubbed GA-FreeCell, solved 98% of Microsoft 32K, thus outperforming HSDH, the (now) previously top solver, which solved only 96% of Microsoft 32K. The average number of nodes, solution time and solution length were 230K, 2.9 and 151, respectively. Thus reducing the amount of search, solution time and solution length by 87%, 93% and 41%, respectively, compared to HSDH. Note that although GA-FreeCell solves "only" 2% more instances, these 2% are far harder to solve due to the long tail of the learning curve.

We are currently performing experiments on the sliding-tile puzzle, with promising initial results: Our best evolved solver outperforms all the solvers but one presented in (Arfaee, Zilles, and Holte 2010), in terms of nodes and time. Our solver solved 1000 test instances with an average number of nodes of 4,991 and average solution time of 0.03. However, the solution lengths are longer, and we assume this is a consequence of using weighted A* with large weights. Our current experiments, which we hope to report upon in the near future, use a variety of weights.

References

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2010. Bootstrap learning of heuristic functions. In *In Proceedings of the 3rd International Symposium on Combinatorial Search (SoCS2010)*, 52–59.

Burke, E. K.; Hyde, M.; Kendall, G.; Ochoa, G.; Ozcan, E.; and Woodward, J. R. 2010. A classification of hyper-heuristic approaches. In Gendreau, M., and Potvin, J., eds., *Handbook of Meta-Heuristics 2nd Edition*. Springer. 449–468.

Elyasaf, A.; Hauptman, A.; and Sipper, M. 2011. GA-FreeCell: Evolving solvers for the game of FreeCell. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011)*. ACM. (to appear).

Hauptman, A.; Elyasaf, A.; Sipper, M.; and Karmon, A. 2009. GP-Rush: using genetic programming to evolve solvers for the Rush Hour puzzle. In *GECCO'09: Proceedings of 11th Annual Conference on Genetic and Evolutionary Computation Conference*, 955–962. New York, NY, USA: ACM.

Heineman, G. T. 2009. Algorithm to solve FreeCell solitaire games. http://broadcast.oreilly.com/2009/01/january-column-graph-algorithm.html. Blog column associated with the book "Algorithms in a Nutshell book," by G. T. Heineman, G. Pollice, and S. Selkow, O'Reilly Media, 2008.

Hillis, D. W. 1990. Co-evolving parasites improve simulated evolution in an optimization procedure. *Physica D* 42:228–234.

Pearl, J. 1984. *Heuristics*. Reading, Massachusetts: Addison–Wesley.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In Fox, D., and Gomes, C. P., eds., *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 357–362. AAAI Press.