Evolving Hyper Heuristic-Based Solvers for Rush Hour and FreeCell

Ami Hauptman, Achiya Elyasaf [*], Moshe Sipper
Dept. of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel
{amihau,achiya.e,sipper}@gmail.com

Abstract

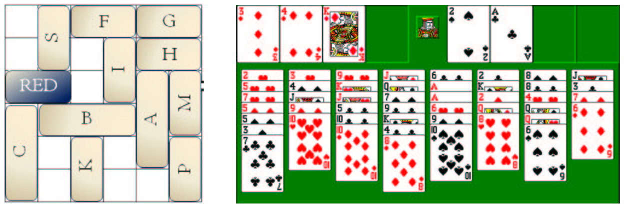
We use genetic programming to evolve highly successful solvers for two puzzles: Rush Hour and FreeCell.

Many NP-Complete puzzles have remained relatively neglected by researchers (see (Kendall, Parkes, and Spoerer 2008) for a review). Among these difficult games we find the Rush Hour puzzle, which was proven to be PSPACE-Complete for the general $n \times n$ case (Flake and Baum 2002). The commercial version of this popular single-player game is played on a 6x6 grid, simulating a parking lot replete with several cars and trucks. The goal is to find a sequence of legal vehicular moves that ultimately clears the way for the red target car, allowing it to exit the lot through a tile that marks the exit (Figure 1a). Another well-known, highly popular example within the domain of discrete puzzles is the card game of FreeCell. Starting with all cards randomly divided into *k* piles, the objective of the game is to move all cards onto four different *foundation piles*—one per suit—arranged upwards from the ace until the king (Figure 1b). FreeCell was proven by Helmert to be NP-Complete (Helmert 2003).

We used genetic programming (GP) to evolve hyper heuristic-based solvers for both Rush Hour and Freecell. Our evolutionary algorithm has proven immensely efficacious, managing to combine heuristics of highly variable utility into composites that are nearly always beneficial, and far better than each separate component.

To date, no efficient heuristics have been reported for the Rush Hour puzzle. Due to the specific structure of the puzzle, standard methods for deriving heuristics, such as solving either sub-problems (possibly with pattern databases (Felner, Korf, and Hanan 2004)), or relaxed problems (e.g., using the Manhattan distance heuristic, augmented with linear conflicts (Hansson, Mayer, and Moti Yung 1992)), which are typically easy to apply to other well-known domains, are not applicable here.

Rush Hour (standard edition) is shipped along with 40 problems, which we designate $JAM01 \ldots JAM40$. To add harder problems to the test suite we expanded it with the



(a) (b)

Figure 1: Sample Rush Hour (a) and FreeCell (b) configurations.

200 most difficult configurations published online by Colette *et al.*, designated $SER1 \ldots SER200$. In order to work with more challenging problems, we also evolved 15 difficult solvable problem on larger, *8x8* boards.

We next describe some of the heuristics we devised, all of which were used to estimate the distance to the goal from a given board. For Rush Hour, the first obvious estimate to the closeness of a board configuration to the goal is the number of vehicles blocking the red car's path to the exit, because when this number reaches zero, the problem is solved. This heuristic is referred to as $BlockersLowerBound$. The next heuristic, called $GoalDistance$, is a possible way to implement the Manhattan-Distance heuristic, as used for the sliding-tiles puzzle (e.g., (Korf and Felner 2002)). The $Hybrid$ heuristic combines the essence of the previous two heuristics: Instead of merely summing up each vehicle's distance to its location in the deduced goal, we also count the number of vehicles in its path, and add it to the sum. Additional functions were used to assign scores to boards, including: $MoveFreed$—which checks if the last move made increases the number of vehicles free to move, and $IsMoveToSecluded$—which checks if the last move placed a car in a position to which no other car can move (Hauptman et al. 2009). Note that the latter functions are not heuristics in the strict sense, in that they do not compute an estimated distance to the goal.

Examples of some of the heuristics for the game of Freecell are: $NumberWellPlaced$—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. The number of cards that are not at the foundation piles is another

[*]Achiya Elyasaf is partially supported by the Lynn and William Frankel Center for Computer Sciences.

heuristic, dubbed $NumCardsNotAtFoundations$. The $LowestHomeCard$ heuristic computes the value of the highest possible card value (typically the king) minus the lowest card value in foundation piles.

Using such heuristics and auxiliary functions to render search more efficient is a difficult task, as it involves solving two major sub-problems: 1) Finding exact conditions regarding *when* to apply each heuristic, and 2) combining several estimates to get a more accurate one.

As we want to embody both application conditions and combinations of estimates, we decided to evolve ordered sets of control rules, or *policies*. The function set included the functions $\{AND,OR,\leq,\geq\}$ for condition trees and the functions $\{\times,+\}$ for the value trees. We used the standard crossover and mutation operators, as detailed in (Koza 1994). However, before selecting the crossover or mutation point, we first randomly selected rules whose conditions (or values) were to be substituted. Fitness scores were obtained by performing full IDA* search, with the given individual used as the heuristic function. For each solved board, we assigned to the individual a score equal to the percentage of nodes reduced, compared to searching with no heuristics. For unsolved boards, the score was 0.

For Rush Hour, not only did we evolve solutions to hard 6x6 boards, we also evolved hard-to-solve 8x8 boards. The most difficult 8x8 board found required 26,000,000 nodes to solve with no-heuristic, iterative deepening.

Results for Rush Hour are summarized in Table 1. Overall, evolved policies managed to cut the amount of search required to 40% for 6x6 boards and to 10% for 8x8 boards, compared to iterative deepening. We also compared the time required to solve these 40 problems by humans to the runtime of several algorithms: iterative deepening, Hi (representing the average time of our three hand-crafted heuristics), our hand-crafted policy, and our best evolved policy (Hauptman et al. 2009): All algorithms tested are much faster than human players, and evolved policies are the fastest. Evolved policies thus save *both* search time *and* space.

Despite its popularity (Bacchus 2001), and despite there being numerous FreeCell solvers available via the Web, few have been written up in the scientific literature. The best solver to date is that of (Heineman 2009), able to solve 96% of the *Microsoft 32K* problem suite—a set of 32,000 problems included by Microsoft in Windows 95. Heineman's algorithm is a hybrid A* / hill-climbing search algorithm called *Staged Deepening* (referred to herein as *HSD*). We used a 2-second time limit per problem instance, under which limitation the HSD algorithm solves 88.7% of the Microsoft 32K.

We designed a number of evolutionary algorithms to seek solvers for FreeCell. The top performer that emerged was coevolution-based GP with policies, which is superior to HSD in several aspects (Table 2): amount of search, time, solution length, and number of solved instances.

Table 1: Average percentage of nodes required to solve $test$ problems, with respect to the number of nodes scanned by iterative deepening (shown as 100% in the second column). $H1$: the heuristic function $BlockersLowerBound$; $H2$: $GoalDistance$; $H3$: $Hybrid$. Hc is our hand-crafted policy devised by combining the basic heuristics, and GP is the best evolved policy.

Heuristic:	None	$H1$	$H2$	$H3$	Hc	GP
Problem						
6x6	100%	72%	94%	102%	70%	40%
8x8	100%	69%	75%	70%	50%	10%

Table 2: HSD vs. our top algorithm—coevolution-based GP with policies. The first three columns show average amount of search, time, and solution length, with HSD serving as a baseline (i.e., 100%). The rightmost column indicates the percent of solved problems of the Microsoft 32K, with a 2-second time limit.

Algorithm	$Amount$ $of Search$	$Time$	$Solution$ $length$	$Solved$ $Problems$
HSD	100%	100%	100%	89%
Evolved Policy	41%	31%	70%	99%

References

Bacchus, F. 2001. AIPS'00 planning competition. *AI Magazine* 22(1):47–56.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive pattern database heuristics. *J. Artif. Intell. Res. (JAIR)* 22:279–318.

Flake, G. W., and Baum, E. B. 2002. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". *Theor. Comput. Sci.* 270(1-2):895–911.

Hansson, O.; Mayer, A.; and Moti Yung. 1992. Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences* 63(3):207–227.

Hauptman, A.; Elyasaf, A.; Sipper, M.; and Karmon, A. 2009. GP-Rush: Using genetic programming to evolve solvers for the Rush Hour puzzle. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*, 955–962. 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.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence* 143(2):219–262.

Kendall, G.; Parkes, A.; and Spoerer, K. 2008. A survey of NP-complete puzzles. *International Computer Games Association Journal (ICGA)* 31:13–34.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1-2):9–22.

Koza, J. R. 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts: MIT Press.