

Search in the Patience Game ‘Black Hole’

Ian Gent₁, Christopher Jefferson₂, Inês Lynce₃, Ian Miguel₁,
Peter Nightingale₁, Barbara Smith₄, Armagan Tarim₂

1: School of Computer Science, University of St Andrews, Scotland. {ipg, ianm, pn}@dcs.st-and.ac.uk

2: Computer Science Department, University of York, England. {chrisj, at}@cs.york.ac.uk

3: IST/INESC-ID, Technical University of Lisbon, Portugal. ines@sat.inesc-id.pt

4: Cork Constraint Computation Centre, University College Cork, Cork, Ireland. bms@4c.ucc.ie

Abstract

We propose card games for one player as a valuable domain for studying search problems. They are a natural AI problem, as they are a widely enjoyed recreation for which solving techniques are generally not studied. We focus on a particular patience, called Black Hole. We show that a general version of it is NP-complete. Then we show that we can fruitfully study a number of mature AI paradigms applied to this single problem. An important feature of Black Hole is the presence of symmetries which arise during the search process, and we show that tacking these can improve search dramatically. Our empirical evaluation shows that Black Hole is winnable approximately 87% of the time.

1 Introduction

We propose patience games – card games for one player also known as “solitaire” – as a fruitful domain for studying search problems. These games are a natural Artificial Intelligence problem, since they are a recreation many people enjoy, but for which solving techniques are generally not studied. There are hundreds of different patiences, with many more variants derived by changing the number of piles or other features. The varied nature of these patiences will lead to different approaches being needed, and this study should help us explore the range of validity of different AI techniques. In particular, as we show here, we can study a variety of mature AI paradigms applied to a single problem. A particular benefit to empirical analysis is that the problems of everyday concern are of a shuffled deck and are therefore entirely random.

There is very little research on patiences: how to solve them, how to play them, and how winnable they are. The only body of work we know of is on the game Freecell. Exactly one of the 32,000 possible games in the original Windows program is unwinnable. Extensive empirical work has shown that the probability of the game being winnable is roughly 99.999%, and solvers are available for the game [Fish,]. Freecell has also been used as a benchmark for AI planning programs [Helmert, 2003]. General solvers could be very useful to players to detect insolubility or give hints. AI research can also feed back by helping design satisfying new

patiences or variants of old ones, for example verifying that a patience is winnable a reasonable percentage of the time.

The variety of patiences is likely to lead to a toolbox of techniques being required instead of a single one. For example, some games reveal all cards at the start and are open to analysis, while others enforce moves to be made with many cards remaining hidden. In other games, in an initial phase cards are placed before a final analytical stage where the cards placed in the first phase are played to a win (or not.) Such different games are likely to be tackled in different ways.

In this paper we show the value of a patiences as a benchmark problem with a case study of the game Black Hole. We solve it using a variety of different AI paradigms, namely planning, constraint programming, SAT, mixed integer programming, and a special-purpose solver. We thus compare the advantages and disadvantages with respect to each other, while also being able to see the important features in each method. Black Hole is particularly appropriate for such a case study. It gives perfect information to the player at the start. Every successful game involves exactly 52 moves, making it easy to apply techniques such as constraint programming to it. As we show here, a natural generalisation of it is NP-complete and therefore we do not expect any shortcuts to be discovered to allow trivial solving in general. Finally, we report that on the problem of human interest, i.e. with 52 cards, Black Hole provides a challenge for all our methods.

We find some paradigms more effective than others in this paper, but absolutely do not claim this shows the more successful techniques are better, even for Black Hole: our (relative) failures may simply be due to a lack of skill and ingenuity on our part. Instead, we intend our analysis to be a useful longitudinal study of a number of AI paradigms on a simple, but not trivial, problem of real interest to people. We emphasise the design decisions in each case and how they relate to the properties of the solvers used and the patience itself, proposing reasons for the techniques’ success or failure.

2 Black Hole

Black Hole was invented by David Parlett with these rules:

“*Layout* Put the Ace of spaces in the middle of the board as the base or ‘black hole’. Deal all the other cards face up in seventeen fans [i.e. piles] of three, orbiting the black hole.

“*Object* To build the whole pack into a single suite based on the black hole.

where there is at least one of each left in the non-literal fans. Finally any remaining *sf* cards are taken using the fan of 3s provided for this purpose.

Lemma 4: *If there is a solution to Black Hole instance generated from a SAT instance, the SAT instance is satisfiable.*

If there is a solution to the Black Hole instance then by Lemma 1, the initial part of this solution must contain exactly one of the cards vT or vF for each variable v . These are a solution as the fans with these cards must between them contain at least one occurrence of c_i for all i , else the solution would be unable to reach the first occurrence of 1.

Theorem 1: *Black Hole is NP-complete.*

Lemmas 3 and 4 show that a SAT instance is satisfiable if and only if its Black Hole encoding is satisfiable, hence the encoding is correct. The encoding is polynomial (in fact linear in the number of literals) therefore Black Hole is NP-hard. Later, we provide a polynomial reduction of Black Hole into SAT, therefore Black Hole is NP-complete.

While this proof has considered encoding a general SAT instance, it could of course encode specialisations of SAT, in particular 3-SAT with a maximum of 5 occurrences of each variable, which is itself NP-complete. Encoding this problem would put a fixed limit on the maximum size of the stacks (to 8) and the number of suits (to 4).

4 AI Planning Model

In AI planning, an initial state is gradually transformed into a goal state through the application of plan operators [Allen *et al.*, 1990]. Black Hole can straightforwardly be characterised in this way: the stacks and initial hole card comprise the initial state, the goal state is that all cards are played, and a move is to play a card.

The plan objects in our PDDL [Fox and Long, 2001] encoding are simply the ranks (ace-king) and the suits (spades, clubs, diamonds, hearts). Each card is specified by a combination of rank and suit objects. We take this approach, as opposed to a single object per card, to simplify the description of adjacency. The initial, current and goal states are described using a number of simple propositions. (`hole rank`) indicates the rank of the card currently in the hole — note that it is not necessary to know the suit of the hole card. (`unplayed rank suit`) and (`played rank suit`) are self-explanatory. We use both because some planners do not accept negated preconditions and/or goals. (`top rank suit`) indicates that a particular card is at the top of a stack in the *initial* state. Similarly, (`under rank1 suit1 rank2 suit2`) indicates that, in the initial state, the card denoted by $rank_1\ suit_1$ is underneath that denoted by $rank_2\ suit_2$. We make use of two further variations of `under`, `underSameSuit` and `underSameRank`, since most AI planners forbid more than one parameter associated with an operator from being instantiated to the same plan object. Finally, (`plusone rank1 rank2`) indicates that $rank_1$ is adjacent below $rank_2$. Thirteen such propositions describe the adjacency of the set of ranks.

The decision to use `plusone` necessitates both `PLAY-UP` and `PLAY-DOWN` operators. The simplest case is in playing a card that is on top of a stack initially, since there is no need to check that the card above has been played. For illustration, `PLAY-UP-TOP` is given below.

```
(:action PLAY-UP-TOP
:parameters (?rank ?suit ?hole)
:precondition (and (top ?rank ?suit)
                  (hole ?hole)
                  (plusone ?rank ?suit)
                  (unplayed ?rank ?suit))
:effect (and (not(unplayed ?rank ?suit))
             (played ?rank ?suit)
             (not(hole ?hole))
             (hole ?rank)))
```

The other operators follow the same basic pattern. There are 12 operators in all (6 in either direction). Five of each six deal with playing a card that was not on the top of a stack initially, with the need to avoid instantiating different parameters with the same plan object accounting for the variations.

For the goal, it suffices to say that the bottom card of each stack must be played, since this implies that all cards above must also have been played. This gives a total of 17 goal conditions in the standard game

We experimented with two state-of-the-art AI planning systems, Blackbox 4.2 [Kautz and Selman, 1999] and FF 2.3 [Hoffmann and Nebel, 2001]. A planner incorporating almost-symmetries was not available to us. FF is a forward-chaining heuristic state-space planner that generates heuristics by relaxing the problem and solves using a Graphplan-style algorithm. Preliminary experimentation revealed that, on this encoding, FF performed by far the better. We compare FF with other techniques, empirically, in Section 9.

5 Constraint Programming Model

Constraint Programming (CP) is a powerful method for solving difficult combinatorial problems. Problems are characterised by a set of constraints on a set of decision variables, which solutions must satisfy, then solved by search.

We can represent a solution to the game as a sequence of the 52 cards in the pack, starting with the ace of spades, the sequence representing the order in which the cards will be played into the Black Hole. This makes it easy to devise a basic CP model. In fact, it is a permutation problem [Hnich *et al.*, 2004]: if the cards are numbered 0 (the ace of spades) to 51, the sequence of cards can be represented as a permutation of these numbers. So we can have two sets of dual variables: x_i represents the i th position in the sequence, and its value represents a card; y_j represents a card and its value is the position in the sequence where that card occurs. We have the usual channelling constraints: $x_i = j$ iff $y_j = i$, $0 \leq i, j \leq 51$, efficiently implemented via the specialised “inverse” constraint. We set $x_0 = 0$.

The constraints that a card cannot be played before the card above it, if there is one, has been played are represented by $<$ constraints on the corresponding y_j variables. The constraints that each card must be followed by a card whose value is one higher or one lower are represented by constraints between x_i and x_{i+1} for $0 \leq i < 51$. We use a table constraint for this, i.e. the constraint is specified by a list of allowed pairs of values.

The variables x_0, x_1, \dots, x_{51} are the search variables: the variables y_0, y_1, \dots, y_{51} get assigned by the channelling constraints. The x_i variables are assigned in lexicographic order,

i.e. the sequence of cards is built up consecutively from start to finish. This simple model using only binary constraints models the problem successfully, but in practice search is prohibitive. We need other techniques to make search practical.

We first deal with the conditional symmetry described in Section 2. Recall that in the example the $9\spadesuit$ and the $9\heartsuit$ are interchangeable if both have been played after the cards above them, the $4\diamondsuit$ and $7\heartsuit$, and before the cards immediately below them, $8\spadesuit$ and $5\diamondsuit$. To break this conditional symmetry, we can add the constraint: if $4\diamondsuit < 9\heartsuit$ and $9\spadesuit < 5\diamondsuit$ then $9\spadesuit < 9\heartsuit$. This constraint forces $9\spadesuit$ to be played before $9\heartsuit$ when they are interchangeable. Given any ordering of the occurrences of each value, all constraints of this form can be added, pairwise, before search. This does not change the solutions returned if (as we describe below) the same order of occurrences is preferred by the value ordering heuristic. The constraints are simplified if the preferred card of the pair is at the top of its pile or the other card is at the bottom of its pile, or both. Because the conditional symmetry breaking constraints are designed to respect the value ordering, the solution found is the same as the solution that would be found without the constraints. The constraints simply prevent the search from exploring subtrees that contain no solution. Hence, the number of backtracks with the constraints is guaranteed to be no more than without them. Furthermore, they appear to add little overhead in terms of runtime; they cannot become active until their condition becomes false on backtracking, and they then become simple precedence constraints that are cheap to propagate. Breaking conditional symmetries can make orders of magnitude difference to the search effort and runtime. One instance took 336,321 backtracks and 326s. to solve without them; when they were added, this was reduced to 252 backtracks and 0.66s.

Initially, when a variable x_i is selected for assignment, we simply selected its values in an arbitrary order (spades first in rank order, then hearts and so on.) We then changed the value ordering, so that cards in the top or middle layers are chosen before cards of the same value lower down in the initial piles. This fits with the problem, in that it makes sense to clear off the top layer of cards as quickly as possible. This also is consistent with the conditional symmetry breaking constraints: as long as values in the same layer are considered in the same order by the heuristic as in the constraints, the same solutions will be returned first with or without the conditional symmetry breaking constraints. Since this is a heuristic, it is not guaranteed to reduce search on each individual instance, but overall, it does reduce search by about an order of magnitude.

Our CP model was implemented in ILOG Solver 6, and is compared with other approaches in Section 9.

6 SAT Model

Satisfiability (SAT) is a technique closely related to CP in which the domains of all variables are Boolean and the constraints are expressed in conjunctive normal form. Specialised solvers achieve large efficiency gains by exploiting the simplicity of this specification language. Our SAT model is conceptually similar to the CP model, although additional variables are needed to achieve the same expressiveness.

We have a 52×52 matrix M of variables, where M_{ij} is true if card i is played into the black hole in the j^{th} position. We know in advance that the $A\spadesuit$ is in the first position. The constraint that each card is played exactly once is represented by at-least-one and at-most-one clauses. Also, clauses are added to ensure that each card is followed by a card whose rank is one value higher or lower. When a solution is found, these variables are used to obtain the solution.

In addition, a second matrix with the same size establishes the order relations between cards. For establishing the order relations, we use a *ladder* matrix [Gent and Prosser, 2002; Ansótegui and Manyá, 2004] i.e. a matrix where for each row we must have a sequence of zero or more true assignments, and all following variables are assigned false. The first entry to have value false gives the position where the respective card has been played. Observe that the entries in this matrix are easily related with the entries in the first matrix. Besides the clauses to guarantee that only valid assignments are allowed, on this matrix clauses are added to guarantee that a card is played into the black hole only after all the cards above it have been played.

Finally, a third *ladder* matrix is required for applying symmetry breaking, where in this matrix the columns contain a sequence of zero or more true assignments, followed by all variables being assigned false. Clauses are placed on this matrix to eliminate conditional symmetries, i.e. search states where cards of the same rank are interchangeable. The conditions under which these symmetries arise have already been described for the CP model.

Unlike CP solvers, most SAT solvers do not provide the option of specifying a variable or value ordering and therefore this part of the CP encoding does not transfer to SAT.

An empirical evaluation of our SAT encoding is in Section 9.

7 Integer Programming Model

Integer programming (IP) is a powerful tool for solving linear optimization problems with discrete decision variables. It has been applied successfully to a variety of fixed-length AI planning problems (e.g. [Jefferson *et al.*, 2005]). We present a binary integer programming model for Black Hole.

In representing an instance with 17 stacks of 3 cards each, the top cards are numbered 1 to 17, the middle cards from 18 to 34, and the bottom cards from 35 to 51. Notation is as follows: $a_{i,j}$ denotes a binary decision variable matrix of size 51×51 , in which the cell (i, j) is 1 if the card at position i is played in move j , otherwise 0; v_i , a parameter array whose i th element denotes the value of the card in position i .

The model below has two sets of constraints: a card may only be played after those above it have been played, and successively played cards must differ in value by either 1 or 12. Eqs. (1)–(3) imply that card i can be played in move j only if the cards above it have been played. Eqs. (4) and (5) guarantee that only one card is played in each move, and each card is played only once. Eq. (6) exploits the fact that, at any move, one can infer infeasible card values for the next 11 moves. Consider Table 1, in which gray cells are feasible card values at the beginning of the game. Table 1 can be generalised to

any move and for any card value. S_j denotes the set of feasible card values in move j . In Eq.(6), the first summation term gives all binary decision variables referring to playing card valued i in move j . The second includes all $a_{k,j+m}$ that don't comply with having a card valued i in move j . If any of the variables specified in the first summation takes on the value of 1, then all $a_{k,j+m}$ in the second must be 0.

$$\sum_{k=1,\dots,j} a_{i,k} \geq a_{i+17,j+1} \quad i = 1, \dots, 17; j = 1, \dots, 50 \quad (1)$$

$$\sum_{k=1,\dots,j} a_{i,k} \geq a_{i+34,j+2} \quad i = 1, \dots, 17; j = 1, \dots, 49 \quad (2)$$

$$\sum_{k=1,\dots,j} a_{i,k} \geq a_{i+17,j+1} \quad i = 18, \dots, 34; j = 2, \dots, 50 \quad (3)$$

$$\sum_{k=1,\dots,51} a_{i,k} = 1 \quad i = 1, \dots, 51 \quad (4)$$

$$\sum_{i=1,\dots,51} a_{i,k} = 1 \quad k = 1, \dots, 51 \quad (5)$$

$$\sum_{\substack{k \in \{1,\dots,51\} \\ v_k = i}} a_{k,j} + \sum_{\substack{k \in \{1,\dots,51\} \\ |v_k - i| \neq \{h-1\} | h \in S_j}} a_{k,j+m} \leq 1 \quad (6)$$

$$j = 1, \dots, 51, \quad i \in S_j, \quad m = 1, \dots, \min\{11, 51 - j\}$$

Black Hole has no objective function, so we employ an artificial one, based upon a breadth-first strategy. This is because breadth-first is expected to yield a feasible solution easily. One such objective function is $\min \sum_{i=1,\dots,17} A \cdot i \cdot x[i] + \sum_{i=18,\dots,34} i \cdot x[i]$, where A is a large constant: penalties for playing top row cards at a later stage in the game are higher, whereas there is no such penalty for bottom row cards. Preliminary experimentation confirmed that this objective function performed better than any other we considered.

Card Value	Moves												
	0	1	2	3	4	5	6	7	8	9	10	11	12
A													
2													
3													
4													
5													
6													
7													
8													
9													
10													
J													
Q													
K													

Table 1: Feasible card values for the beginning of the game

Experiments were performed on a 2GHz PC using Ilog Cplex 9.0. Cplex's emphasis indicator was set to "emphasis feasibility over optimality". The test suite consisted of 30 randomly-generated instances. The solution time was limited to 5 hours. In three cases there were no feasible solutions, which was proven at the root nodes in a few seconds. Of the remaining instances, only 20 were solved in the time limit. The shortest solution time was 140s (solved at the root node); the longest was 17000 sec, with 2203 nodes visited.

These results suggest that IP is not an effective approach to Black Hole. This is somewhat surprising, since, as noted, IP

has been successfully applied to other fixed-length AI planning problems. The lack of a real objective function to provide a tight linear relaxation is certainly a factor, as well as the fact that the linear encoding of Black Hole requires a very large number of binary variables and constraints.

8 Special Purpose Solver for Black Hole

The advantage of a special-purpose solver is that, knowing the properties of the problem, code can be optimised to search exceptionally fast. The disadvantage is the lack of mature and deep techniques for search, or the difficulty of adapting and implementing these techniques for the domain.

The solver is written in Common Lisp. To avoid garbage collection, no lists were constructed or discarded after initialisation at the root of the search tree. The other key design principle was to minimise the amount of work on making moves and undoing them for backtracking.

Essentially we treat cards as pointers into the data structures. Each card is represented as an array index, so cards are numbered from 0 to $suits * ranks - 1$. For each card, we construct at the root a static pointer to the card immediately above it in its pile, or a null pointer if it is on top. A simple dynamic bitarray indicates whether each card has been played or not at this point in search, leading to one bit change on moving forward and backtracking. A card is available if it has not been played, but the card above it (if any) has been. Our data structures make this a very cheap test. We simply test all possible cards, of the ranks one above and below the hole card, to see if they are available. The list of cards of each rank is computed statically at the root. We set up a 2-D array to store available moves at each depth, the dimensions being depth in search and $2 \times$ the number of suits. At a new depth we insert the possible moves into this array. No work in this array needs to be done on backtracking, except decrementing the pointer to the current depth.

We deal with conditional symmetries as follows. First, we distinguish between 'unit' cards, i.e. cards at the bottom of a pile or above only cards of the same rank, and other cards that we call 'general'. A card's unity or generality is determined statically at the root, so lists of general and unit cards are stored at the root. When finding playable cards, we consider at most one unit card of each rank, and none at all if there are general cards of that rank. This deals with this limited kind of conditional symmetry almost without overhead. Our solver described until now is able to search at almost 100,000 backtracks per second on a 2GHz PC. However, we are prone to exceptionally hard problems: one winnable instance took 1,055,774,437 backtracks and 11,701s.

We adapted the solver to deal with general conditional symmetry, i.e. when two general cards of the same rank are simultaneously available. The overheads are now more substantial, and we did not find great reductions in search from dealing with conditional symmetry, so overall performance was not dramatically improved.

To conclude, we were able to write a special-purpose solver which could search very fast, but its lack of reasoning abilities means that the cost in larger search spaces is not repaid by the added speed per node compared to other methods. Overcom-

ing this problem would improve runtimes but at a substantial overhead in programmer time.

9 Experimental Evaluation

We have reported five solvers in this paper. The MIP approach and the special purpose solver were not competitive with the others, with many instances taking hours to solve. This is in no sense a final conclusion that these approaches are impractical for Black Hole solving, but we restrict their empirical evaluation to the brief details reported above. We have performed a much more extensive empirical comparison of our AI Planning, CP, and SAT based solvers.

We constructed a single benchmark set of 2,500 instances. All three solvers were tested on the same instances, and gave the same results on each instance – excepting a few timeouts described below. Since the instances were randomly generated instances of the standard game with 52 cards, we can report on the expected winnability of Black Hole. A total of 2,189 instances were winnable and 311 were not, giving an 87.56% probability of winnability. The 95% confidence interval for the true probability is [86.2%, 88.8%].

We were not able to run the three remaining solvers on the same machines, but we have normalized runtime results as if they were. We take the CP solver to have a factor of 1. To derive the runtime multipliers we ran a single SAT solver on the same set of benchmarks on the three different machines. We used “siege” as our SAT solver [Ryan, 2004]: since this has randomised features we ran each instance 50 times with different seeds, and report the mean value of these.

All three solvers were highly effective at solving these instances. Only the FF planner failed to solve all instances in less than 2,800s CPU time, and it solved all but 46, i.e. more than 98% of instances. Of those it did solve, the longest took only 3 minutes to solve. For CP, the longest time was 1,454s, and for SAT the longest mean time was 180.9s.

This suggests a rank order of SAT, CP, FF. However, investigating percentiles of behaviour is more interesting. The best median is CP, at only 0.04s, then FF at 0.73 and SAT at 2.2s. While FF continues to be dominated by CP, we see SAT starting to outperform CP at the higher percentiles. SAT solves 97.5% of instances in 17.0s, compared to 20.8 for CP, and for the highest percentiles the gap widens considerably. Even this hides considerable complexity, as the two solvers find different instances difficult. There is only a weak correlation between the difficulty experienced by SAT and CP solvers, $r = 0.22$.

Despite the different profiles, mean performance of SAT and CP solvers is very close, at 3.92s and 4.35s. A paired t-test on the two data sets does not reject the null hypothesis that performance is the same ($p = 0.62$.) However, given our study of the profiles, it is reasonable to say that our CP solver is usually quicker, while the SAT solver is more robust at solving all instances in reasonable time. The CP solver solves 84% of instances in less than 1s, while the SAT solver solves *all* instances in a mean of less than 3 minutes and 1s.

To conclude, we found that while all solvers could solve Black Hole instances, the SAT and CP solvers were the most effective and hardly distinguishable. FF was just behind those two, with our other two approaches not nearly as successful.

10 Conclusions

Our first contribution has been to understand the game Black Hole, proving it NP-complete in general, and showing that the original game has a winnability percentage of about 87%.

Next, this has been a case study to compare different modelling techniques using five AI paradigms. We do not wish to say that one technique is better than another, but make some general points. The most natural technique for the problem, planning, did have the most natural model with least subtle points in implementation. Even here, the choice of planner used was critical to our success, so one cannot say this is an easy problem for planning. Also, raw speed is not overwhelming in this domain: the special purpose solver was the fastest solver in terms of backtracks per second, but thrashed too much to solve most instances rapidly. Our paper also serves as a case-study of the use of conditional- or almost-symmetries. These proved essential in the CP solver and, where it was not available to us (in the planning solver), might have hindered our success. Despite the maturity of these research areas, this illustrates the necessity of constant research to improve performance.

Finally, we propose that patience games provide a fecund area for investigation. Many interesting questions of many different types can be asked, and answered, to the benefit of the AI community.

References

- [Allen *et al.*, 1990] J. Allen, J. Hendler, and A. Tate. *Readings in Planning*. Morgan Kaufmann, 1990.
- [Ansótegui and Manyá, 2004] C. Ansótegui and F. Manyá. Mapping problems with finite-domain variables into problems with boolean variables. In *SAT 2004*, 2004.
- [Fish,] S. Fish. Freecell solver. <http://tinyurl.com/4jt7q>
- [Fox and Long, 2001] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains, 2001.
- [Fox and Long, 2002] M. Fox and D. Long. Extending the exploitation of symmetries in planning. *AIPS 2002*, pages 83–91, 2002.
- [Gent and Prosser, 2002] I. P. Gent and P. Prosser. SAT encodings of the stable marriage problem with ties and incomplete lists. In *SAT 2002*, 2002.
- [Gent *et al.*, 2003] I. P. Gent, I. McDonald, and B. M. Smith. Conditional symmetry in the all-interval series problem. In *Proc. SymCon'03*, pages 55–65, 2003.
- [Helmert, 2003] M. Helmert. Complexity results for standard benchmark domains in planning. *Artif. Intell.*, 143(2):219–262, 2003.
- [Hnich *et al.*, 2004] B. Hnich, B. M. Smith, and T. Walsh. Models of permutation and injection problems. *JAIR*, 21:357–391, 2004.
- [Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.
- [Jefferson *et al.*, 2005] C. Jefferson, A. Miguel, I. Miguel, and A. Tarim. Modelling and solving english peg solitaire. *Computers and Operations Research (in press)*, 2005.
- [Kautz and Selman, 1999] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proc. IJCAI-99*, 1999.
- [Parlett, 1980] David Parlett. *The Penguin Book of Patience*. Penguin, 1980.
- [Ryan, 2004] L. Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.