Incremental Transpositions

Bernard Helmstetter, Tristan Cazenave

Université Paris 8, laboratoire d'Intelligence Artificielle
2, rue de la Liberté 93526 Saint-Denis Cedex France
{bh, cazenave}@ai.univ-paris8.fr

Abstract. We introduce a distinction, in single-agent problems, between transpositions that are due to permutations of commutative moves and transpositions that are not. We show a simple modification of a depth-first search algorithm which can detect the transpositions of the first class without the use of a transposition table. It works by maintaining, for each node, a list of moves that are known to lead to transpositions. This algorithm is applied to two one player games: a solitary card game called *Gaps*, and a game called *Morpion Solitaire*. We analyze, for each domain, how often transpositions are due to commutative moves. In one variant of Gaps, the algorithm enables to search more efficiently with a small transposition table. In Morpion Solitaire, a transposition table is not even needed. The best known sequence for this game is proved optimal for more than one hundred moves.

1 Introduction

In games, transpositions often arise because the same moves appear in two sequences but in different orders. For example, when two moves a and b are commutative, the sequences (a, b) and (b, a) lead to the same position. This is not the only possibility; for instance, if m and m^{-1} are inverse moves, then the sequence (m, m^{-1}) leads to the same position as the null sequence.

We state that we do not need a transposition table to detect transpositions of the first class. Instead, they can be detected by maintaining incrementally, for each node of the game tree, a list of the moves that are known to lead to transpositions.

We present an algorithm that implements this idea, called the incremental transpositions algorithm. When a search is made with a small transposition table, comparatively to the real size of the tree, this may result in a dramatic increase of the number of nodes searched. We show that our algorithm can help attenuate this phenomenon.

The incremental transpositions algorithm is presented in section 2. Section 3 presents its application to a solitary card game called *Gaps*; we analyze two variants of this game. In section 4, we introduce a game called *Morpion Solitaire*, which is a perfect application domain since all transpositions can be detected without a transposition table.

2 The incremental transpositions algorithm

We present a modification of a depth-first search algorithm that recognizes all transpositions that are due to permutations of commutative moves, and we show that it is complete. Finally, we discuss the conditions for applicability of the algorithm.

2.1 Algorithm

The function, called `dfs_it` and shown below, takes as argument a set of moves T which are known to lead to transpositions, that is to say positions that have already been completely searched. The function is to be called at the root position with $T = \emptyset$.

function dfs_it(T) {
  for each legal move m, $m \notin T$,
    $T' = \{ t \in T$, t is legal after m, m is legal after t,
      t and m are commutative$\}$;
    do_move(m);
    dfs_it(T');
    undo_move(m);
    $T = T \cup \{m\}$;
}

The set T' is built so that it is restricted to moves that will be legal after the current move m; therefore, for all recursive calls to `dfs_it`, the argument set T will be a subset of the set of legal moves. Consequently, this algorithm needs only very little memory.

2.2 Proof of completeness

We state that the incremental transpositions search algorithm is complete. We have to show that, supposing the set T given as parameter to the function `dfs_it` contains only moves which lead to positions that have already been completely searched, the same property holds for the recursive calls of `dfs_it` on the child nodes.

Let m be a move at the current node a and $t \in T$, such that t is legal after m, m is legal if played after t, and the moves t and m are commutative. This means that we have the commutative diagram shown in figure 1. At node a, we know that move t leads to a node, c, that has already been completely searched. Therefore node d, which can be reached from node b with move t, has also been completely searched. Since node d can also be reached from node c with move m, we conclude that move t at node b leads to a transposition.

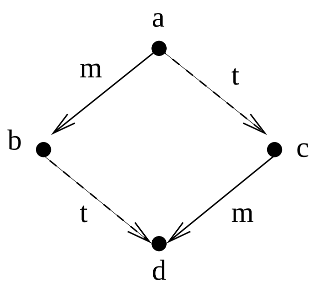


Fig. 1. Transposition

2.3 Applicability of the algorithm

The incremental transpositions algorithm is applicable whenever the game contains commutative moves. It is therefore broadly applicable. However, since it does not detect all transpositions, it should be viewed as a complement to a transposition table rather than a substitution to it. This being said, the question is to assess what can be gained from the algorithm.

First, what is the proportion of transpositions that are due to permutations of commutative moves? This question is very domain-dependent. For example, there are absolutely no commutative moves in the sliding-tile puzzle. In Rubik's cube, there are few commutative moves (only moves on opposite faces are commutative), so the algorithm would not be very useful. The two domains that we will study, Gaps and Morpion Solitaire, feature a large number of commutative moves; in the second case, we will show that all transpositions are due to permutations of commutative moves. We have also been able to apply the algorithm with some success in Sokoban.

Secondly, the incremental transpositions algorithm is only interesting for large searches, when a transposition table alone is not enough. In chess, experiments by D. Breuker [Bre98] show that a transposition table can detect most transpositions even when the size of the transposition table is much less than the size of the search space. One of his experiments shows that, for a search that would take optimally about 100×10^6 nodes, the situation is already almost optimal with a transposition table of size 1M positions, and we lose less than a factor two when using a transposition table of size 8K. This situation, however, doesn't appear to be general. Our experiments with Gaps will show that the number of nodes searched explodes when the transposition table is smaller than the size of the search space.

Finally, our algorithm only works for a depth-first search in one player games. It would be difficult to apply it to two-player games, because it is not sufficient to know that a move leads to a transposition, it is also necessary to know the value of the position after the move. This value has to be stored in memory.

3 Gaps

We describe the rules of two variants of the game called *Gaps* or *Montana*. We give examples of non-incremental transpositions for both variants, and we give experimental results of the incremental transpositions algorithm for one of them. We compare three possibilities: transposition table alone, incremental transpositions alone, and both at the same time.

3.1 Rules

We first describe the variant we call the basic one. The game is played with a 52-card deck. The cards are placed in 4 rows of 13 cards each. The 4 Aces are removed, resulting in 4 gaps in the position; then they are placed in a new column at the left in a fixed order (*e.g.*, 1st row Spade, 2nd Heart, 3rd Diamond, 4th Club). The goal is to create ordered sequences of the same suit, from Ace to King, in each row. A move consists in moving a card to a gap, thus moving the gap where that card was. A gap can be filled only with the successor of the card on the left (that is, the card of the same color and one higher in value), provided that there is no gap on the left and that the card on the left is not a King, in which case we can place no card in that gap. Figure 2 shows an initial position with only 4 cards per suit, before and after moving the Aces, and the possible moves.

4♠	3♣	4♣	4♡
2♢	4♢	2♡	2♣
2♠	3♢	1♡	1♣
3♠	1♢	3♡	1♠

1♢	4♠	3♣	4♣	4♡
1♡	2♢	4♢	2♡	2♣
1♠	2♠	3♢	□	□
1♣	3♠	□	3♡	□

Fig. 2. An initial position with 4x4 cards, before and after moving the Aces. This position can be won.

We will be concerned with another variant of the game which we call the common one, since it is the one most often played. In this variant, the Aces are not placed in a column on the left of the first column but are definitely removed. Instead, it is allowed to place any Two in a gap if it is on the first column. As we will see, this rule is the cause of many transpositions that are not due to permutations of moves.

3.2 Previous work

T. Shintani has worked on a game he calls *Superpuzz*; it is actually another name of Gaps, precisely what we call the common variant [Shi00]. He has analyzed the structure of the strongly connected components in the search graph. This is complementary to our research in this paper since, as we will see, the moves inside the strongly connected components are the cause of many transpositions that are not due to permutations of moves.

In [HC03], it is shown how the game can be decomposed in relatively independent subgames, and the dependencies between them are analyzed. An algorithm called *blocks-search* has been designed, which works by grouping several positions in one block and making a search on the blocks rather than on the positions alone. As a side effect (it was not the primary objective of the algorithm), it was observed that blocks-search could detect more transpositions when the size of the search space exceeds the size of the transposition table. The incremental transpositions algorithm presented here is simpler; it does not do simplification of the search space as blocks-search does, but it does a better job to detect the transpositions. Up to now, attempts to combine both algorithms have not been successful.

3.3 Examples of non-incremental transpositions

We show two examples of transpositions that are not due to permutations of moves. Although there are other possibilities, it gives a rough idea of the structure of the transpositions in the two variants of Gaps introduced.

The first example, shown in figure 3, occurs in both variants. The sequences m_1, m_2 and m_3, m_1, m_4 give the same position. It may be possible to make the algorithm more efficient by adding a rule to automatically detect this pattern, and thus increase the number of transpositions detected without a transposition table.

The second example, shown if figure 4 (a), is specific to the common variant. The position after moves m_1 and m_2 is the same as the position after move m_3. The position shown in figure 4 (b), with many gaps on the first column and long ordered sequences on the right, features many similar transpositions. Therefore, the incremental transpositions algorithm is not well suited to these kinds of positions. Here again, it would probably be possible to add rules to detect those patterns, such as preventing the same card from moving twice in a row. Also, the methods developed by T. Shintani [Shi00] seem to be particularly efficient precisely in these kinds of positions.

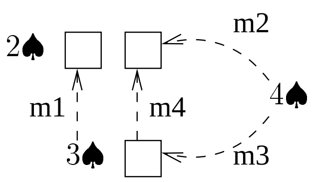


Fig. 3. example of transposition occurring in both variants

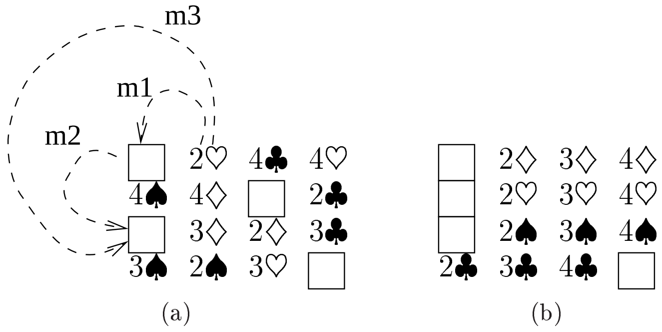


Fig. 4. examples of transpositions occurring only in the common variant

Transpositions of the kind of the second example appear more often than those of the kind of the first example. Therefore, the proportion of transpositions that are not due to permutations of moves is larger for the common variant than for the basic one. The next section shows experimental results for the basic variant.

3.4 Experimental results for the basic variant

The experimental results have been made on the basic variant with 52 cards. We are doing a complete search and we do not stop after a winning sequence has been found. The game could always be searched completely with a depth-first search and a transposition table, for all the random positions we have tested so far.

The transposition table has been implemented with a hashtable of a fixed number of entries, and one position per entry. When collisions happen, the old position is replaced with the new. More complex replacement schemes with two positions per entry may be slightly more efficient [Bre98].

First, we analyze how much we lose by using incremental transpositions alone, rather than combining it with a transposition table. We compare the number of nodes searched using a transposition table and incremental transpositions, $N_{tt\_it\_32M}$, with the number of nodes searched using incremental transpositions alone, N_{it}. Figure 5 shows the ratio $N_{it}/N_{tt\_it\_32M}$ depending on $N_{tt\_it\_32M}$, for 1000 random initial positions. Here the size of the transposition table is $2^{25} =$ 32M entries, more than the number of positions searched, so $N_{tt\_it\_32M}$ is very close to the real size of the search space when all transpositions are detected. The good point of this experiment is that we manage to make a complete search without a transposition table, although the cost in number of nodes relatively to the optimal is between a factor 1 and 1000.

Secondly, we show the usefulness of combining incremental transpositions with a limited size transposition table. Here we work with a transposition table of a reduced size: $2^{20} =$ 1M entries. We compare the number $N_{tt\_it\_1M}$, with the number of nodes searched when using a transposition table alone, N_{tt_1M}. Figure 6 shows the ratio $N_{tt\_1M}/N_{tt\_it\_1M}$ depending on $N_{tt\_it\_1M}$, for 3200 random initial positions. This shows how useful the incremental transpositions algorithm can be, in conjunction with a transposition table. When the size of the search space exceeds the size of the transposition table, the number of positions searched when using only a transposition table explodes.

Another fact that can be seen on this second diagram is the large increase of nodes searched when using a limited size transposition table. This is a loss compared to $N_{tt\_it\_1M}$, so *a fortiori* it is a loss compared to the optimal. This increase is particularly sensitive for problems of size more than 2×10^6 nodes.

Our third experiment shows that this phenomenon is much weaker when the transposition table is combined with the incremental transpositions algorithm. We compare the number of nodes searched for two different sizes of the transposition table: 1M and 32M. We call these two numbers $N_{tt\_it\_1M}$ and $N_{tt\_it\_32M}$. Figure 7 shows the ratio $N_{tt\_it\_1M}/N_{tt\_it\_32M}$, depending on $N_{tt\_it\_32M}$, for 5000 random initial positions. The diagram shows that, when the transposition table is backed up with the incremental transpositions algorithm, it is much less affected by the size of the search space.

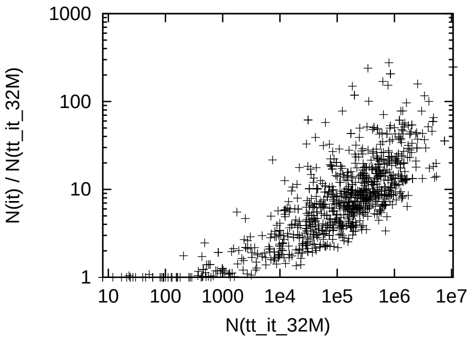


Fig. 5. Comparison between transposition table (32M entries) combined with incremental transpositions, and incremental transpositions alone

The experiments have been run on a Pentium 3GHz with 1GB memory. Our implementation of the depth-first search, without a transposition table and incremental transpositions, can search 20.4×10^6 positions/s. This drops down to 9.5×10^6 positions/s with incremental transpositions, and 1.7×10^6 with a transposition table. With both, the search is actually faster than with a transposition table alone: 3.4×10^6 positions/s. The reason is that incremental transpositions are tested before accessing the transposition table; our implementation of the incremental transpositions algorithm is very well optimized and can detect transpositions faster than the transposition table lookup.

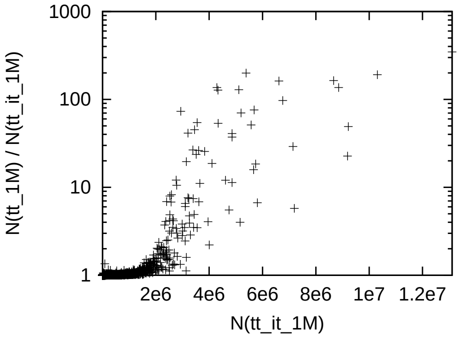


Fig. 6. Comparison between transposition table (1M entries) combined with incremental transpositions, and transposition table alone

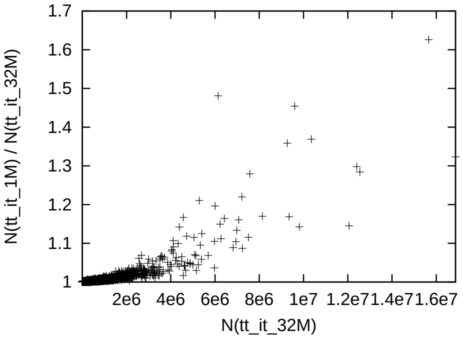


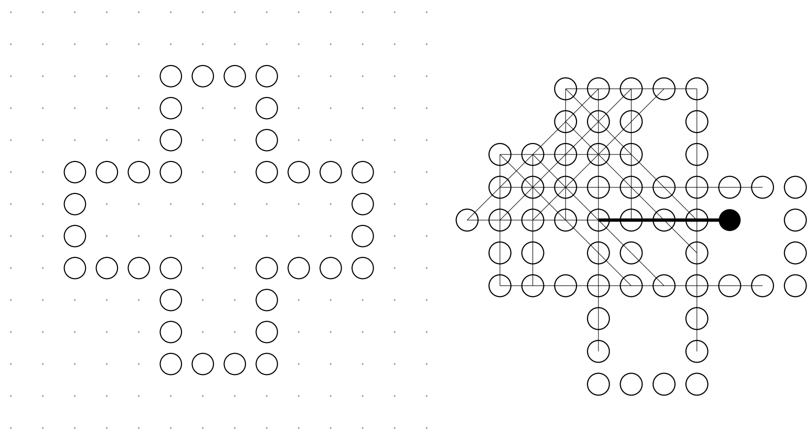
Fig. 7. Transposition table combined with incremental transpositions, with transposition table size 32M and 1M

4 Morpion Solitaire

We introduce a relatively new game called Morpion Solitaire. We show that, when doing a complete depth-first search of the game tree, using the incremental transpositions algorithm is sufficient to detect all transpositions, and therefore a transposition table is useless. Although we don't beat the current record, we are able to prove its optimality for more than one hundred moves.

4.1 Rules

Morpion Solitaire is a one player deterministic game, with a fixed starting position. General information about the game can be found in [Mey]. It is played using a paper with a grid layout, ideally infinite, and a pencil. The initial position, with 28 dots, is shown in figure 8 (a). A dot can be added if it makes an alignment of five dots, horizontally, vertically or diagonally, with the dots already drawn. The move consists in adding the dot and the segment of line, of length four, that goes through the five dots. Additionally, an alignment cannot be drawn if it shares more than one dot along the same axis as any previous line segment. Figure 8 (b) shows an example of a legal move. The goal is to maximize the number of moves.



(a) (b)

Fig. 8. positions at the beginning and after a few moves

4.2 Bound on the length of the game

It has been proved that the length of the game is bounded. We explain the argument briefly. Each dot has 8 possibilities to be used for drawing an alignment, one for each direction. Some of them may have already been used. The sum, for each dot, of the number of free possibilities, is an invariant during the game. Indeed, when we make a move, this number is increased by 8 because we add a new dot, and decreased by 8 because of the segment of line drawn: the 3 dots inside the alignment lose 2 possibilities each , and the 2 dots at the extremities lose 1 each. The more dots are drawn, the more there are dots "on the boundary" (dots that are not completely surrounded by other dots). Those have free possibilities, so the length of the game is bounded. Achim Flammenkamp has proved a bound of 324 moves [Mey].

4.3 Finding long sequences: previous work

The current best published record for the game is 170; it has been obtained by hand, by Charles-Henri Bruneau, in the 1970's [Mey].

Hugues Juillé has applied a method which he calls Evolving Non Determinism to the game [Jui95,Jui99]. This algorithm works by making a population of prefix sequences evolve; the best ones are selected depending on the results of random games starting with the prefixes; the prefix sequences are gradually made longer until the end of the game is reached. The best result obtained by Juillé with this algorithm is 122. Pascal Zimmer has later improved the algorithm and found a sequence of length 147 [Mey].

Although this is not the subject of this paper, we have ourself developed a program to find good sequences. This has been done independently with Juillé's work, but the method is similar, since it also makes heavy use of statistics on random games. Our current record is 136.

In the following, our goal will be to determine if the record, of length 170, is optimal in the endgame, starting from as as many moves as possible before the end.

4.4 All transpositions are due to permutations of moves

We show that, in Morpion Solitaire, all transpositions are due to permutations of commutative moves. First we remark that, whenever two moves a and b are legal as well as the sequences (a, b) and (b, a), they are always commutative. Therefore, all we need to show is that, given a position, we can find the set of moves that have been made to reach it. This is done in two steps.

First, we group all the segments of line drawn in segments of length four, each being the effect of only one move. We can do it separately for all the segments lying on the same infinite line, and in this case it is easy: we just have to group them four by four in order.

Secondly, we find the exact position where a dot has been added on each of the segments of line just found. We start from the initial position, and, whenever a move is possible that corresponds to one of the segments of length four found, we make it. We lose nothing by doing such a move as soon as it can be done. On one hand, no other move will ever be possible using this segment of length four even after other moves have been made; on the other hand, doing this move now or later doesn't change the possibilities for the other moves.

We have thus shown theoretically that the transpositions are due to permutations of moves; this is confirmed experimentally by the fact that, whether we make a depth-first search with incremental transpositions and a transposition table or incremental transpositions alone, the same number of nodes are searched.

4.5 Experimental results

We have made experiments to test if the best known sequence, of length 170, is optimal for the last moves. We have also made a similar search for the best sequence we have found ourself, of length 136.

Since we don't need a transposition table, we have been able to parallelize the search. This search has run for about two months on the computers of our lab. Our program can search 3.7×10^6 positions/s on a Pentium 3GHz.

Figure 9 (a) shows the evolution of the total number of nodes searched depending of the depth of the starting position. Starting at move 61, a total of 3.97×10^{13} nodes have been searched, and the record has not been beaten. Figure 10 shows the positions after move 61 and after the last move for the best known sequence.

Figure 9 (b) shows the same work for the best sequence found by ourself (section 4.3). We have put less effort on this one, but we have still been able to search it completely starting at move 42. This sequence has not been improved either.

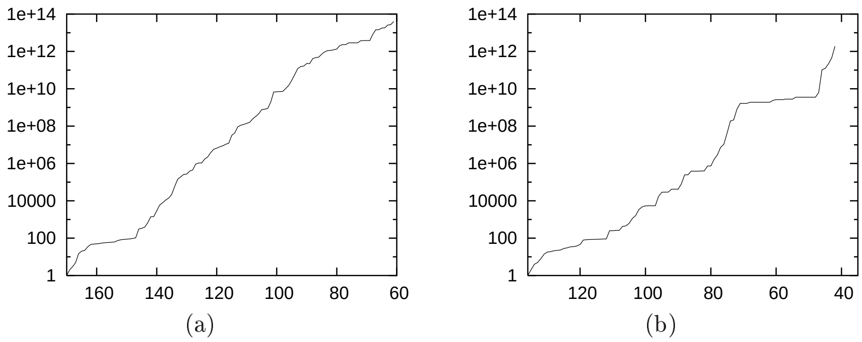


Fig. 9. number of positions searched depending on the starting move (a) in the best known sequence of length 170, (b) in a sequence of length 136

The fact that the sequences considered in this section are optimal for so many moves is surprising, especially for the record of length 170 which has been obtained by hand. In our opinion, this indicates that the endgame is relatively easy. Both human players, and computers with the methods described in section 4.3, are able to play it well.

5 Conclusion

We have introduced a distinction between transpositions that are due to permutations of moves and transpositions that are not. The first category of transpositions can be easily detected using a method we have called incremental transpositions. Unlike a transposition table, this method costs very little memory. It does not seem, however, to be applicable in two players games.

The proportion of transpositions that are due to permutation of moves, and therefore the efficiency of the algorithm, depends on the domain. In the basic variant of Gaps, the algorithm can be used in conjunction with a transposition table to detect more transpositions. Results are particularly good when

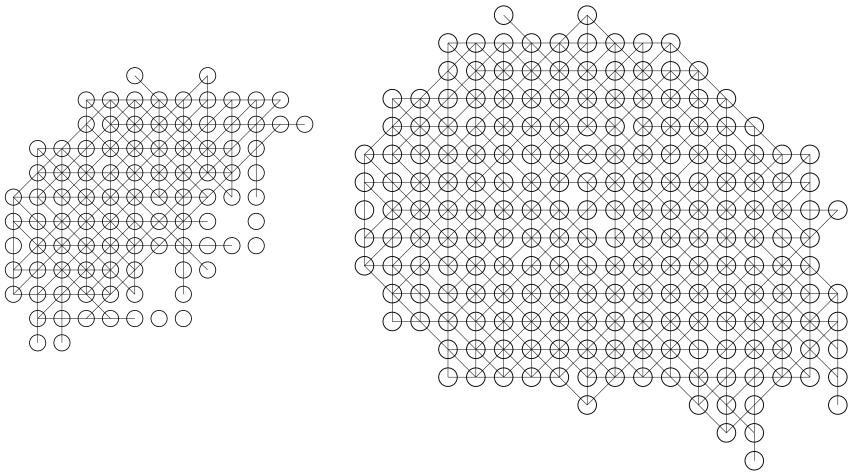


Fig. 10. Positions after move 61 and after move 170

the search space is larger than the size of the transposition table. The method would be less efficient in the common variant. Finally, we have shown a perfect application domain for the algorithm, Morpion Solitaire, where the incremental transpositions algorithm makes a transposition table useless.

In the domain of Morpion Solitaire, our parallel search program has proved the optimality of the best known sequence for more than one hundred moves before the end. We know that methods using statistics on random games are efficient to find the best sequences in the endgame. Therefore, as further research, we plan to make use of statistics on random games to make cuts in the search, so that we can test the optimality for more moves before the end, and still be quite sure of the result.

References

[Bre98] Dennis M. Breuker. *Memory versus Search in Games.* PhD thesis, Maastricht University, 1998.

[HC03] Bernard Helmstetter and Tristan Cazenave. Searching with analysis of dependencies in a solitaire card game. In *Advances in Computer Games 10*, pages 343–360, 2003.

[Jui95] Hugues Juillé. Incremental co-evolution of organisms: A new approach for optimization and discovery of strategies. In *European Conference on Artificial Life*, pages 246–260, 1995.

[Jui99] Hugues Juillé. *Methods for Statistical Inference: Extending the Evolutionary Computation Paradigm.* PhD thesis, Brandeis University, 1999.

[Mey] Jean-Charles Meyrignac. Morpion solitaire progress. http://euler.free.fr/morpion.htm.

[Shi00] Toshio Shintani. Consideration about state transition in Superpuzz. In *Information Processing Society of Japan Meeting*, pages 41–48, Shonan, Japan, 2000. (In Japanese).