More Games of No Chance
MSRI Publications
Volume **42**, 2002

Global Threats in Combinatorial Games: A Computation Model with Applications to Chess Endgames

FABIAN MÄSER

ABSTRACT. The end of play in combinatorial games is determined by the normal termination rule: A player unable to move loses. We examine combinatorial games that contain *global threats*. In sums of such games, a move in a component game can lead to an immediate overall win in the sum of all component games. We show how to model global threats in Combinatorial Game Theory with the help of infinite loopy games. Further, we present an algorithm that avoids computing with infinite game values by cutting off branches of the game tree that lead to global wins. We apply this algorithm to combinatorial chess endgames as introduced by Elkies [4] where this approach allows to deal with positions that contain entailing moves such as captures and threats to capture. As a result, we present a calculator that computes combinatorial values of certain pawn positions which allow the application of Combinatorial Game Theory.

1. Global Wins and Global Threats

Combinatorial game theory (CGT) applies the divide and conquer paradigm to game analysis and game tree search. We decompose a game into independent components (local games) and compute its value as the sum of all local games. The end of play in a sum of combinatorial games is determined by the normal termination rule: A player unable to move loses. Thus, in a sum of games, no single move or game can be decisive by itself. We investigate a class of games where a move in a local game may lead to an overall win in the sum of all local games. We call such a move *globally winning.*

Examples of globally winning moves are moves that capture a vital opponent piece such as *checkmate*,[1] moves that promote a piece to a much more powerful one like promoting a king in *checkers*, or moves that "escape" in games where one side has to try to catch the other side's pieces like in the game *Fox and Geese*

[1] Although checkmate does not actually capture the enemy king, it creates the unstoppable threat to do so.

(*Winning Ways* [2], chapter 20). Figure 1 shows a Fox and Geese position where the fox escapes with his last move from e5 to d4 and obtains "an infinite number of free moves". If this game were a component of a sum game S, the fox side would never lose in S.

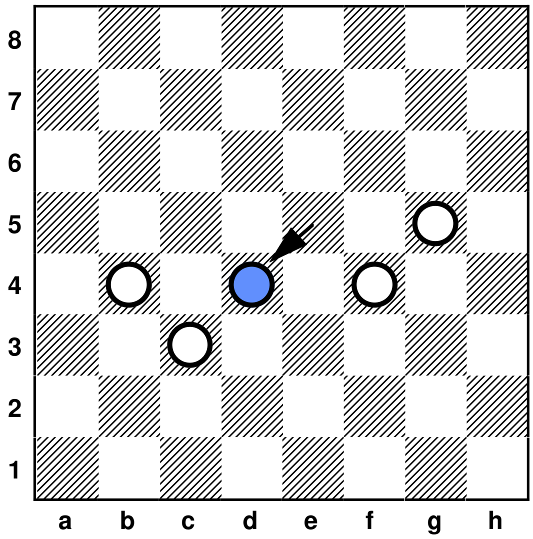

Figure 1. A position in the game *Fox and Geese*: the last move by the (dark) fox *escapes* the geese that are only allowed to move upwards.

We are mainly interested in games where none of the players can win by executing a global threat if the opponent defends optimally. Such games are finally decided by normal termination and have finite combinatorial values. However, when we search the tree of all moves in order to compute a game's value, we also have to execute and undo globally winning moves. In section 2, with the help of infinite games, we model the situation that arises after one of the players has executed a globally winning move. A global win is equivalent to having an infinite number of moves available. In section 3, we present an algorithm that computes combinatorial values of games with global threats. In order to avoid computing with infinite game values, the algorithm cuts of the game tree just before a globally winning move is executed. Section 4 applies these techniques to king and pawn endgames in chess.

Regarding chess endgames, Elkies [4] writes: "The analysis of such positions is complicated by the possibility of pawn trades which involve entailing moves: an attacked pawn must in general be immediately defended, and a pawn capture parried at once with a recapture. Still we can assign standard CGT values to many positions . . . in which each entailing line is dominated by a non-entailing one." This paper introduces an algorithm that solves this problem in general.

2. A CGT Model Based on Loopy Games

A natural way to model global wins in combinatorial game theory is to define values G_{lwin} and G_{rwin}. The game G_{lwin} stands for the situation that *Left* has executed a globally winning move. It must be greater than any finite game G. Analogously, G_{rwin} stands for the situation that *Right* has executed a globally winning move and has to be smaller than any finite game G.

$$\forall \text{ finite games } G : G_{rwin} < G < G_{lwin} \tag{2–1}$$

The simplest games (they consist of only one position) that satisfy (2–1) are the loopy games $on = \{on \mid \}$ and $off = \{ \mid off\}$ (*Winning Ways* [2], chapter 11). The game *on* is greater than any *ender*[2] G. *Left* wins any difference (*on* - G) by just playing in *on* until *Right* runs out of moves in $(-G)$.

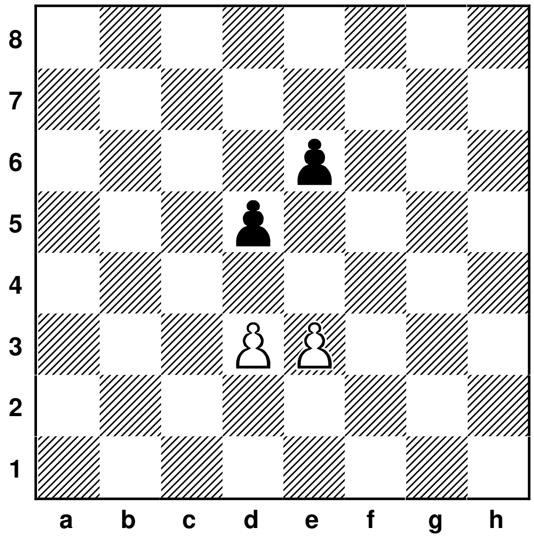

Figure 2. Global threats in a chess position: Both players have to prevent the opponent from promoting a pawn to a queen. Play will end when the pawns are blocked and neither player has any moves left.

Figure 2 shows a chess example. In the context of king and pawn endgames, we consider promoting a pawn to be globally winning.[3] For instance, White's move e3-e4 leads to a symmetrical position where both players have the choice either to capture or to push the more advanced pawn. We compute its value as $\{*, \{on \mid *\} \mid *, \{* \mid off\}\}$.

[2]a game in which no player can play an infinite sequence of moves.

[3]This is true for a majority of pawn endgames, and we limit our attention to these.

As the values *on* and *off* only appear as *threats*, the resulting game value of Figure 2 is finite. None of the players is able to force the promotion of a pawn if his opponent plays correctly. Here is a short analysis of the game position: White's move d3-d4 and Black's move e6-e5 both lead to zero positions. It turns out that both other moves are reversible (White's e3-e4 and Black's d5-d4), and thus the game value is $G = \{0 \mid 0\} = *$.

If both players have the possibility to force a global win in different subgames, the result of the sum is a "draw by repetition". In the sum $(on + off)$, both players will always have moves available, therefore none will lose:

$$on + off = \{on + off \mid on + off\} = dud$$

The result *dud* ("deathless universal draw" [2]) offers a pass move for both players, so any sum S containing at least one component game of value *dud* will never be brought to an end $(dud + G = dud)$.

3. An Algorithm Based on Cutoffs in the Game Tree

While the model presented in section 2 works fine in theory, it is not so easy to implement in practice. We usually map finite combinatorial games "one to one" to data structures by their inductive definition $G = \{G^L \mid G^R\}$ with the basis $0 = \{\,\mid\,\}$. Following a path of left and right options, we are sure that finally the zero game will be reached. Almost all algorithms that work on combinatorial games are based on their inductive nature. Obviously, loopy games do not fit into this model. Either, we must extend the model to handle loopy games, or we must somehow avoid loopy games in our computations.

In this section, we pursue the second way. We present a computation model for local games with global threats that cuts off branches of the game tree that lead to global wins. The model is based on the following two lemmas:

(i) If a player has the chance to execute a globally winning move, he will always do so.
(ii) Any move to a position which offers the opponent a globally winning move is "bad". Such a move need not be considered when evaluating a player's options.

Both lemmas follow directly from the rules for *simplifying games* (see [3] or [2]). The first one is easily deduced: As for any game G, the equation $off \leq G \leq on$ applies, a move to a globally winning position always *dominates* any other possible move. The second one is deduced from the rule of replacing reversible moves. If in the game $G = \{G^L \mid G^R\}$, *Left* plays a move to G^{L_i} which contains a right move to *off*, then *Left's* move to G^{L_i} is *reversible* $(G \geq off)$ and is replaced by all left options of *off*. As *off* has no left options, *Left's* move to G^{L_i} is simply omitted. The same holds of course for a right move to a game G^{R_j} that offers a left move to *on*.

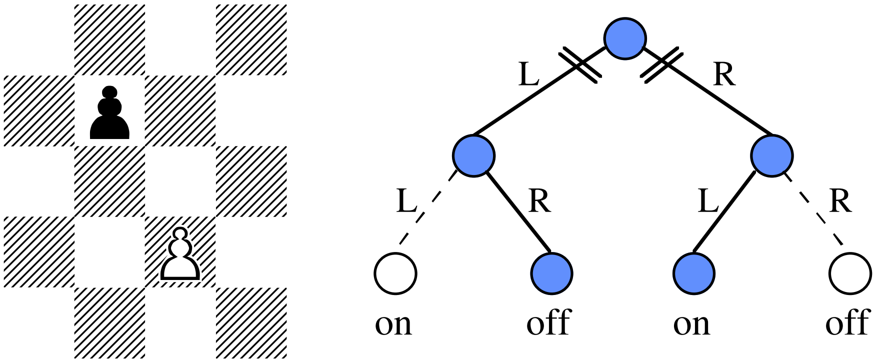

Figure 3. A game of value zero and its game tree. Each player's move leads to an immediate opponent win. This is equivalent to having no moves at all. $G = \{\{on \mid off\} \mid \{on \mid off\}\} = \{ \mid \} = 0$

The chess position on the left side of Figure 3 illustrates this. Both players have exactly one move which leads to a position where the two pawns attack each other. The player who captures his opponent's pawn will go on to promote his pawn to a queen. The value of this position is computed as $G = \{\{on \mid off\} \mid \{on \mid off\}\}$ which simplifies to $\{ \mid \} = 0$.

Applying the model presented in section 2, we compute the value of G by producing its game tree up to the terminal positions *on* and *off*. Instead, based on the second lemma, we can immediately cut off both players' moves as we already know that they will be reversible (see the right side of Figure 3). This leads directly to the same value $G = \{ \mid \} = 0$. There are two evident advantages of this approach. First, we avoid most calculations with loopy games. In this simple example, we do not have to deal with loopy games at all. Second, thanks to the cutoffs, we minimize the number of nodes to be searched in the game tree.

Now we are ready to formulate an algorithm for evaluating local games with global threats. We consider both players' options in each position. Additionally, however, we make use of the information who played the last move. This allows us to cut off moves that lead to global wins for the opponent. It might seem unusual to make use of to-play-information in combinatorial game tree search, but this also occurs implicitly in conventional CGT. The same rule of replacing reversible options that allows us to cut off the game tree is based on "good replies" to an opponent's move, thus also makes use of to-play-information.

3.1. Result Types. In contrast to finite combinatorial games, the value of a game that contains global threats might be *on* or *off* i.e. a forced global win for one of the players. This is the case if a player cannot prevent his opponent from finally playing a globally winning move no matter how he defends. In order to separate finite combinatorial values from global wins, we distinguish the following *result types* of local games:

- Type *win*: If the players move alternately including the right to pass, *Left* will win by executing a globally winning move no matter how *Right* defends and no matter who starts.
- Type *loss*: If the players move alternately including the right to pass, *Right* will win by executing a globally winning move no matter how *Left* defends and no matter who starts.
- Type *CGT*: None of the players can force a win by global threat. In this case, a combinatorial value $G = \{G^L \mid G^R\}$ can be computed for the actual game position.

3.2. Game Tree Search. In order to evaluate a game, we produce its tree starting from the current position. In every position, we recursively evaluate both players' options. As the possible result types are ordered ($win > CGT > loss$ from *Left's* point of view and $loss > CGT > win$ from *Right's* point of view), we can determine the best result types both players can get if they have the move. If a player's best result type is *CGT*, we also compute his combinatorial game options (G^L respectively G^R). Combined with the information on who is to play, we compute the result type of the actual position as shown in Figure 4. In case the actual result type is *CGT*, we also compute the combinatorial game value of the actual position.

R / L	**loss**	**CGT**	**win**
loss	loss	CGT_1	CGT_2
CGT	loss / CGT	CGT_3	CGT_4
win	loss / win	CGT / win	win

Figure 4. Result Types of *Global Threats Evaluation*: the table indicates the result type of a local game depending on the best result types of *Left's* (rows) and *Right's* (columns) options and on the right to make the next move. The split entries show the result types for *Left* to play (lower left) and *Right* to play (upper right).

In the four cases labeled *CGT*, we can compute a finite combinatorial value for the actual game position.

(i) All *left* options lead to games of type *loss* while *Right* has at least one move that leads to a finite combinatorial game. As *Left* has no good moves, the value of the actual position is $G = \{\ \mid G^R\}$.

(ii) Neither player has any good moves. The position is a *mutual Zugzwang*. $G = \{\ |\ \} = 0$. We have already seen an example of such a situation in Figure 3.
(iii) Both players' best options all lead to finite combinatorial values. The actual game value is computed as $G = \{G^L \mid G^R\}$.
(iv) In contrast to the first case, *Right* has no good move while *Left* has at least one CGT option. $G = \{G^L \mid\ \}$.

If both players' best result types are *win* (resp. *loss*), the result type of the actual game position is of course also *win* (resp. *loss*). In the remaining three cases, the result type of the actual position will depend on who has the right to move. If the player to move can move to a winning position, he will of course do so and the result type is determined as a win in his favor. Of special interest are the positions where the player to move has one or more moves that lead to games of type CGT while his opponent would be winning if he was to play. These are the only cases where the loopy games *on* and *off* occur in our combinatorial game values which are either $\{on \mid G^R\}$ or $\{G^L \mid off\}$). Fortunately, *on* and *off* only appear as *threats*. For example, *Left* plays a move to a position of value $\{on \mid G^R\}$ when *Right* immediately has to move to one of the options in G^R as *Left threatened* to move to *on*.

3.3. Implementation. The function *GTSearch* searches the game tree depth first and computes result types and combinatorial game values of local games that contain global threats. Its specification is

- **in**:
  - *toplay*: the player (constants *kWhite*, *kBlack*, *kNoPlayer*) who has the move. In the starting position, *kNoPlayer* is passed. After at least one move is played, the right to move is determined.
- **out**:
  - *return value*: the result type (constants *kWin*, *kLoss* or *kCGT*) of the current position.
  - *value*: the combinatorial game value of the current position. This value is "valid" only in case *kCGT* is returned.

The algorithm performs the following steps (numbers refer to comments in the code):

(i) Check termination:
determine if the actual position is a global win for one of the players. If so, we are finished and return *kWin* resp. *kLoss*.
(ii) Recursively evaluate *Left's* options:
We store the best result type (*kWin* > *kCGT* > *kLoss*) achieved so far in the variable *bestL*. For every option of type *kCGT*, we include its combinatorial

value in the set G^L. If we find an option leading to result type $kWin$, we can skip the remaining options (cutoff!).

(iii) Recursively evaluate *Right's* options:
As in step 2, we compute the values *bestR* and G^R.

(iv) Compose the result:
According to the table in Figure 4, we compute the result type of the actual position. In case the result type is $kCGT$, we also compute the combinatorial game value $\{G^L \mid G^R\}$ and return it in the out-parameter *value*.

```
function GTSearch(toplay: TPlayer; var value: TGameValue): TResultType;
begin
  if GlobalWin(kLeft) then return kWin; endif; /* 1 */
  if GlobalWin(kRight) then return kLoss; endif;
  G^L ← { }; bestL ← kLoss;
  forall left moves m do /* 2 */
    ExecMove(m);
    res ← GTSearch(kRight, val);
    UndoLastMove();
    if res = kWin then bestL ← kWin; break endif; /* cutoff! */
    if res = kCGT then bestL ← kCGT; G^L ← G^L∪ val endif;
  endfor;
  G^R ← { }; bestR ← kWin;
  forall right moves m do /* 3 */
    ExecMove(m);
    res ← GTSearch(kLeft, val);
    UndoLastMove();
    if res = kLoss then bestR ← kLoss; break endif; /* cutoff! */
    if res = kCGT then bestR ← kCGT; G^R ← G^R∪ val endif;
  endfor;
  return ComposeResult(bestL, bestR, value, G^L, G^R, toplay); /* 4 */
end GTSearch;
```

4. Application to Chess Endgames

Elkies [4] has shown that certain types of chess endgames (mostly king and pawn endgames) can be analyzed using Combinatorial Game Theory. If both players' kings (or any other remaining pieces) are bound by *mutual Zugzwang* (mZZ), the remaining pawn moves (*tempi* in the chess literature) decide who will have to give way. Thus, when we identify a mZZ, we try to decompose the position and calculate local values for the independent pawn chunks. Figure 5 shows a position from an actual game (Sveda - Sika, Brno 1929) which has been solved by Elkies. Its value is $\uparrow$ (queenside, a and b files) plus 0 (center, a mZZ involving both kings) plus $\downarrow\downarrow *$ which adds up to $\downarrow *$, a first player win.

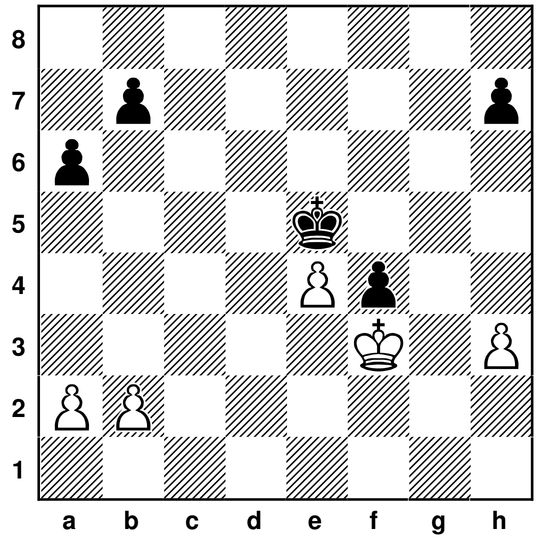

Figure 5. Mutual Zugzwang: Sveda - Sika, Brno 1929 The player who first has to move his king loses. Therefore the player who makes the last pawn move in the sum game of the queenside and kingside pawns will win.

Position with pawns on one file only, like the kingside in Figure 5, are easy to analyze. The pawns will get blocked in any possible line of play. Because of the possibility of moving a pawn by two squares from its original square they are not completely trivial. Positions with two or more pawns on each side (for example the queenside in Figure 5) are more complex. The key to the analysis of such structures are the *global threats* of promoting pawns that reach their final rank. With the computation model presented in section 2, we can also handle positions like the one shown in Figure 6.

The kingside with king and pawn each is "the same" mutual Zugzwang that we have already seen in the Sveda-Sika game. The pawn structure on the queenside looks like it will block after a few moves, but in fact thanks to his far advanced pawns white to move can force a global win by sacrificing two pawns in order to promote the third one. After 1.b5–b6 a7×b6 (c7×b6 2.a5–a6 etc.) 2.c5–c6 b7×c6 3.a5–a6 the a-pawn is unstoppable. In the chess literature, this maneuver is known as a *breakthrough*. Black to play, on the other hand, cannot do the same, as white would be much faster promoting one of his own pawns. His only move that does not allow white to win by global threat is 1... b7-b6 leading to a value $G_1^L = \{0, \{on \mid 0\} \mid \mathit{off}\}$.

We conclude that captures and attacks are often *entailing moves* because they usually lead to global threats in form of the promotion of a pawn. In

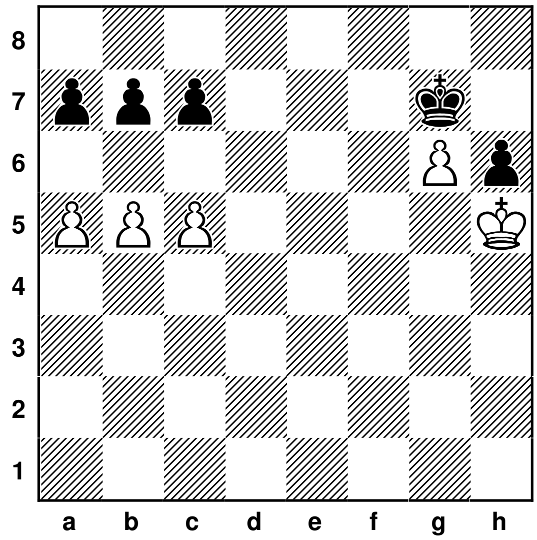

Figure 6. The breakthrough: White to play sacrifices two pawns in order to promote the third one. Black to play loses by Zugzwang.

many pawn structures, however, especially if both sides have an equal number of pawns, none of the players can force a win by promoting a pawn, and we can calculate a combinatorial game value for the given position.

4.1. Implementation. Based on the algorithm presented in section 3, we have implemented a pawn structure calculator within the *Game Bench* [6] project. The Game Bench is an application framework for programs that implement combinatorial games. One of its main goals is to separate algorithms and data structures common to all combinatorial games from the details of specific games and make them available to the game programmer.

The Game Bench is written in Java which makes it portable to almost all of today's computer platforms. Its variety of game independent support makes the Game Bench well suited for "fast prototyping". On the other hand, thanks to "just in time compilation" of Java bytecode, it allows serious game analysis and time critical calculations of combinatorial games as well. Furthermore, on Unix and Windows platforms, David Wolfe's *Gamesman's Toolkit* [7] is used in the form of a dynamic library of efficient C-functions. This library performs all basic CGT computations which are the most time critical part of algorithms like combinatorial game tree search.

4.2. Results and Conclusions. Applying combinatorial game theory to king and pawn chess endgames involves the following three steps:

(i) Detection of *Zugzwang* and identification of the component games.
(ii) Evaluation of the local component games.
(iii) Combination of the local results.

Step one is chess specific, whereas steps two and three are independent of the game we want to analyze. Although we can detect Zugzwang positions involving the kings (and possibly other pieces) automatically, e.g. with the help of a local minimax search, the conclusion that such a Zugzwang will be constant is of a heuristic nature.

Let's again look at the Sveda-Sika game (Figure 5): If White starts, play might continue 1.h3–h4 a6–a5 2.h4–h5 a5–a4 3.h5–h6. Black could now consider to abandon his f4-pawn and instead attack the white pawn on h6. After the moves 3...♔e5–f6 4.♔f3×f4 ♔f6–g6 5.♔f4–e5 ♔g6×h6, material is balanced but white has an easy win with 6.♔e5–f6 as his e-pawn is unstoppable. Thus, our initial assumption that the side to move its king first loses, was correct. But in other cases, breaking out of the Zugzwang might upset the combinatorial evaluation of the position.

Thus, we see the main application of a pawn structure calculator as a tool to support human chess players when analyzing and explaining certain chess endgames. Positions in which the pawns fight for extra tempi are only vaguely described in the chess literature, and a verification of the results of human analysis by brute force minimax search is still beyond the scope of today's leading chess software if the positions are complex enough.

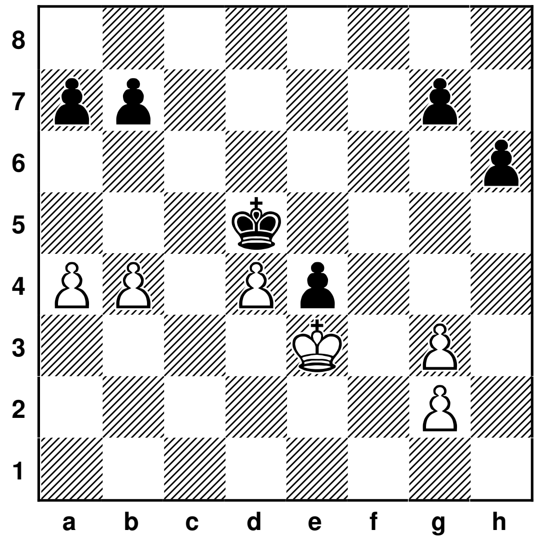

Figure 7. A first player win: Popov - Dankov, Albena 1978. The queenside and center are both of value 0. On the kingside, the first player to move forces his opponent into a zugzwang position. The kingside and therefore the whole game has a value of $G = \{0 \mid -1\}$.

Figure 7 shows a position from the game Popov vs. Dankov (Albena 1978). The relative position of the two kings is the same as in the Sveda-Sika game.

The player to move his king loses his central pawn. Thus, the game is decided by the subgames on the queenside and kingside.

- The queenside is a game of value $G_{QS} = 0$. With the white pawns advanced to the fourth rank, Black gets no advantage from the double step option. The player to move gets blocked immediately.
- The kingside is a bit more complex. Black to play gains a considerable advantage with 1...h6-h5. In fact, due to White's option to sacrifice a pawn for a tempo with g3-g4, it "only" leads to a value of -1. White to play has only one move. Thanks to the possibility of sacrificing one of the doubled pawns for a tempo, it leads to a position of value 0. The main line runs 1.g4 g6 2.g5 h×g5 3.g4 and a zero position with no moves for either side is reached. The value of the kingside is $G_{KS} = \{0 \mid -1\}$.

Thus, the sum $G_{QS} + G_{KS} = \{0 \mid -1\}$ is a first player win. Awerbach [1] assesses this position correctly, but does not offer other calculation methods than brute force search. He write (in German): "With pawns on one or two files, computing spare tempi is not too difficult..." Conventional chess programs, on the other hand, have more problems evaluating this position. The following results are computed on a PC (466 MHz Intel Celeron, 128 MByte Ram) running Linux.

- The CGT algorithm presented in section 3 requires a total of less than 1000 evaluated nodes to compute the values of the kingside and the queenside.
- Combinatorial evaluation of the combined (kingside and queenside) pawn structure yields the same result of course, but takes much longer. Almost 200,000 nodes need to be evaluated.
- In order to illustrate the complexity of a full-width alpha-beta search, we ran Crafty [5] on the game position with White to move. Only after evaluating more than $2 \cdot 10^9$ nodes, the program indicated 1.g3–g4 leading to a white advantage.

5. Summary and Outlook

Games with global threats are an interesting extension of conventional combinatorial games. They model entailing moves such as captures in king and pawn chess endgames. As we can see in Figure 6, the heuristic that a capture should be answered with a recapture fails to produce exact results. We can represent global threats with infinite, loopy games. On the other hand, we propose an algorithm for game tree search that avoids dealing with infinite game values by cutting off branches of the game tree that lead to global wins.

Certain chess endgames allow the application of combinatorial game theory [4]. We decompose the pawn structure into independent local games, calculate their values and finally compute the value of the sum by rules of CGT. However, promoting a pawn to a queen in a local game results in a global win in the sum of all component games. The analysis of local pawn structures becomes very complex if the number of pawns increases. A calculator that computes values of such pawn structures automatically is a useful tool for the game analyst, especially as most game positions are still too complex to be searched at full width by conventional chess programs.

The analysis of pawn structures presented in this paper has potential applications beyond CGT. We have used divide and conquer in combination with CGT to combine the results of local games. But the results computed by the *GT-Search* algorithm can be used in more general settings as well. As an example, the information that a cluster of white and black pawns can generate a passed pawn for one of the players may be used in a heuristic evaluation function that rates pawn structures.

References

[1] J. Awerbach. *Bauernendspiele*. Sportverlag, Berlin, 1983.

[2] E. Berlekamp, J. H. Conway, and R. Guy. *Winning Ways for Your Mathematical Plays*. Academic Press, New York, NY, USA, 1982.

[3] J. H. Conway. *On Numbers and Games*. A K Peters, 2001.

[4] N. D. Elkies. On Numbers and Endgames: Combinatorial Game Theory in Chess Endgames. In R. J. Nowakowski, editor, *Games of No Chance*, pages 135–150. Cambridge University Press, New York, 1996.

[5] R. Hyatt. Crafty v17.9, a very strong freeware chess program, rated 2499 on the Elo scale by the Swedish Chess Computer Association in August 2000.

[6] F. Mäser. *Divide and Conquer in Game Tree Search: Algorithms, Software and Case Studies*. PhD thesis, ETH Zürich, 2001.

[7] D. Wolfe. The Gamesman's Toolkit. In R. J. Nowakowski, editor, *Games of No Chance*, pages 93–98. Cambridge University Press, New York, 1996.

Fabian Mäser
Ergon Informatik AG
Bächtoldstrasse 4
8044 Zürich
Switzerland
fabian.maeser@ergon.ch