A CHESS COMPOSER OF TWO-MOVE MATE PROBLEMS

Fridel Fainshtein [1], *Yaakov HaCohen-Kerner*[12]
Ramat-Gan, Israel Jerusalem, Israel

ABSTRACT

Computerized chess composers of mate problems are rare. Moreover, so far they do not produce either impressive or creative new mate problems. In this paper, we describe a model called CHESS COMPOSER. This model uses a 64-bit representation, an ordered version of Iterative-Deepening Depth-First Search, and a quality function built with the help of two international masters in chess-problem composition. The result is applied on 100 known problems. It shows that the quality of 97 problems has been improved. Some of the improvements are rather impressive considering that most of the tested problems were composed by experienced composers. The new, improved problems can be regarded as creative from the viewpoint of experts in chess compositions, because (1) they seem to be better, and (2) they are not too similar to the original problems.

1. INTRODUCTION

Computer composition of mate problems in chess is a relatively uninvestigated research domain. The few computerized composers of such mate problems do not produce impressive or creative, new mate problems.

Thompson (1986, 1996) and Nalimov (2000) built practical chess endgames databases. Their studies only dealt with 5 -piece and 6-piece endgames. The databases are currently used by many strong chess-playing programs. Other endgame studies were performed by Heinz (1999), Tamplin and Haworth (2001), and Haworth (2003). Among the outcomes, there are two-move mate problems too.

Thompson (1986, 1996) used a retrograde analysis to build his endgame databases. Schlosser (1988, 1991) was probably the first to propose a general method of using a retrograde analysis for the composition of chess problems. His method included three main stages: (1) constructing a complete database of new problems starting from given mating positions [in a similar way to the construction of endgames by Thompson], (2) eliminating all incorrect positions according to chess composition rules, and (3) selecting high-quality chess problems based on evaluation values given by a human chess expert. Watanabe, Iida, and Uiterwijk (2000) implemented similar ideas for composing problems in Tsume-Shogi (a Japanese chess-like game).

The concept of a "high-quality mate problem" in chess is hard to define, especially if an automatic program is involved. It is not simple to define concepts such as beauty, originality, uniqueness of the solution, and difficulty to solve. Therefore, a major part of the knowledge needed for evaluating the quality of chess problems in general and of two-move problems in particular has been defined in a model called an IMPROVER OF CHESS PROBLEMS (ICP) (HaCohen-Kerner, Cohen, and Shasha 1999). This knowledge was collected with the help of two international masters in chess-problem composition. It includes definitions of themes, bonuses, and penalties in the domain of chess composition of two-move problems.

ICP tries to improve the quality of a given problem by a series of transformations. It uses a hill-climbing search, satisfying several chess-composition criteria at each step. ICP improved the quality of 10 out of 36 known two-move mate problems (about 28%). Most of the improvements achieved are considered as slight improvements. The main differences between ICP and Schlosser's model are:

1. ICP uses a move-forward hill-climbing technique from the given problem instead of using a move-backward retrograde analysis;
2. ICP starts with 2-movers and ends with them; there is no change in the number of the moves;
3. ICP uses a computer instead of an expert to evaluate the new problems.

[1]. Department of Computer Sciences, Bar-Ilan University, 52900 Ramat-Gan, Israel, fainshf@cs.biu.ac.il.
[2] Department of Computer Sciences, Jerusalem College of Technology (Machon Lev), 21 Havaad Haleumi St., P.O.B. 16031, 91160 Jerusalem, Israel, kerner@jct.ac.il.

In this article a new model called the CHESS COMPOSER is described. It uses a different algorithm for the composition of high-quality problems. CHESS COMPOSER's database contains 100 problems including the 36 problems examined by ICP. The new model improved 97 out of 100 problems. So it performs much better than ICP (97% vs. 28%).

The course of the article is as follows. Section 2 describes the CHESS COMPOSER. Four illustrative examples are given in Section 3. Section 4 presents the results of experiments and analyzes them. Section 5 summarizes the research and proposes future directions.

2. THE CHESS COMPOSER MODEL

CHESS COMPOSER composes 2-movers by improving previous 2-movers. It uses a fixed-depth brute-force method and a heuristic function that estimates the quality of chess problems.

In order to find the best improvement(s) for each problem (if existing), we investigate all possible closely related positions and compare their quality values to the quality value of the original problem. To generate all closely related positions, we use a brute-force search method. All regular brute-force search methods have a time complexity of $O(b^d)$, where b represents the branching factor and d represents the depth of the generated positions tree. This depth is the number of the applied changes/transformations starting from an original problem. In our domain, b represents all applied transformations rather than all legal chess moves. The estimated average branching factor in the chess game is some 33 to 36 (dependent on the precise definition of mobility, cf. Hartmann, 1989), while our branching factor is 368 on average (see Table 1 in Subsection 2.3).

2.1. General Description

CHESS COMPOSER uses the Iterative-Deepening Depth-First search. (Korf, 1985) proved this scheme to be asymptotically optimal among all brute force methods. The total time for nodes to be expanded is $O(b^d)$, where b is the branching factor, and d is the depth of the developed tree.

Practically, observing a small set of problems, we discovered that a depth of 3 levels would be a normal bound, since it takes about 32 minutes on average for a problem at this level, while it takes about 50 hours at level 4. For problems with no improvement after a sequence of the three transformations we continue researching one level deeper. The general algorithm is described in Figure 1.

The main components
Composer is the main function. It receives as input *MaxDepth* which is equal to 3. Then, it *iterate*s consequently to depth 1, depth 2, and depth 3. If no improvement has been found, it iterates to depth 4.

Iterate is a bounded DFS function. It generates all nodes until *Composer's* bound (level 3). In order not to reapply the heavy function *ImprovedMateProblem,* the algorithm calls this function only for nodes on the last level. By this, we gain some improvement in inner nodes and reduce the main disadvantage of the Iterative-Deepening scheme which may duplicate the expansion of the inner nodes. The Iterative-Deepening scheme, described in Korf (1985), expands inner nodes $MaxDepth - d$ times, $0 < d < MaxDepth$, while in our algorithm these nodes are expanded just once. Of course, the number of generated nodes is the same in both algorithms. However, the number of generated notes is less important because the cost of generating a node is much less than the cost of its expansion.

ImprovedMateProblem checks whether a new position is an improved 2-move mate problem. It is a very costly function because it calls three functions: *LegalChessPosition*, *LegalTwoMoveMateProblem*, and *ProblemEvaluator.* The last function is the heaviest, because it checks whether a new position contains any themes, bonuses, and penalties.

```
Composer (OriginalProblem, MaxDepth)
1. ImprovedList = NULL
2. For ( I = 1; I <= MaxDepth; I ← I + 1)
   2.1.Iterate (OriginalProblem, I)
3. If isEmpty (ImprovedList )
   3.1.Iterate (OriginalProblem, MaxDepth + 1)
4. return bestScored problem from ImprovedList

Iterate (OriginalProblem, Bound)
1. PushIntoStack ({OriginalProblem, 0})      ˜˜ 2nd parameter is for the node's
   level in the tree
2. While   NotEmptyStack ( )
   2.1.   {CurrentPosition, Level}  ←  PopFromStack()
   2.2.    if  Level = Bound     ˜˜.in this case we investigate the CurrentPosition
      2.2.1.    If ImprovedMateProblem (CurrentPosition, OriginalProblem) then
         2.2.1.1. ImprovedList ← StoreImprovedMateProblem (CurrentPosition)
   2.3.    else        // i.e.: if (Level < Bound) ˜.in this case we do not
       investigate the CurrentPosition
      2.3.1.    For each Successor of CurrentPosition // from right to left
        2.3.1.1.    CurrentPosition = ApplyNextTransformation
              (CurrentPosition)
        2.3.1.2.   PushIntoStack ({CurrentPosition, Level + 1})

ImprovedMateProblem (CurrentPosition, OriginalProblem)
1. if    ( LegalChessPosition (CurrentPosition)  and
            LegalTwoMoveMateProblem (CurrentPosition) and
            ProblemEvaluater (CurrentPosition) > ProblemEvaluater
   (OriginalProblem))
   then return true
   else return false
```

Figure 1: Optimized Iterative-Deepening DFS Composer.

Other components

- *LegalChessPosition* checks the legality of a position according to the chess rules.
- *LegalTwoMoveMateProblem* tests whether a given position is a two-move mate problem (including the test for non-cooking, i.e., no more than one key move). This component uses a suitable limited search engine and checks all legal chess moves including the two special moves (in contrast to ICP), *castling* and *en passant* capture.
- *ProblemEvaluator* analyzes a position as a mate problem and computes its quality score. This function is described in the next subsection.
- *ApplyNextTransformation* applies the next transformation on a given position according to a fixed ordered list of transformations.

CHESS COMPOSER uses three kinds of transformations, while attempting to improve a problem (the *ApplyNextTransformation* function): (1) deletion of a specific piece, (2) addition of a specific piece, and (3) transparency of all pieces. The last one is done through two possible movements: (a) file-transparency: all pieces are transferred *i* files to the right-side or to the left-side and (b) rank-transparency: all pieces are transferred *i* ranks to the upper-side or to the lower-side.

Additional possible transformations (e.g., moving a certain piece, exchanging a piece) are introduced in Kerner (1995) and HaCohen-Kerner *et al.* (1999). These transformations can be seen as complex transformations based on the basic transformations: deletions and additions. That is, every transformation can be presented by suitable deletion(s) and addition(s).

2.2 Heuristic Evaluation Function

The quality of a chess problem is not simple to evaluate. There are no exact definitions for concepts such as beauty, originality, uniqueness of the solution, and difficulty to solve. Moreover, it seems that most composition judges do not apply a grid with points when they are judging a chess problem. However, for an automatic program there is a need to define such a function. Therefore, with the help of two international masters in chess-problem composition, we built a heuristic function that computes a quality score of a mate problem.

The following heuristic function is defined in ICP:

$$q_m(problem) = \begin{cases} 0 & \textit{severe deficiency} \\ \sum_i V(T_i) + \sum_j V(B_j) - \sum_k V(P_k) & \textit{otherwise} \end{cases},$$

where q_m is the quality square of the discussed problem. A severe deficiency is considered as a cancellation of a position as a proposed problem. A severe deficiency happens when one of the three following situations occurs: (1) illegal chess position, (2) it is not a two-move mate problem, (3) there is more than one key move (cooked). V is the value function. T_i is the set of all themes (Appendix A) included in the position. *Bj* is the set of all bonuses (Appendix B) granted to the position. P_k is the set of all penalties (Appendix C) granted to the position. Various themes, bonuses, and penalties were collected consulting two international masters in chess composition and taken from the classic books of chess composition (Harley 1970, Howard 1961). Appendixes A to C were presented in HaCohen-Kerner *et al*. (1999). We reproduce them because: (1) some definitions have been revised and (2) for the readers' comfort.

We define and apply eleven composition themes: self-blocking, self-pinning, unpinning, half-pinning, direct battery, indirect battery, Grimshaw, Pickaniny, king-flights, lonely black king, and tempo. Each theme has its own unique score depending on its complexity and relative importance. The definitions of these themes are given in Appendix A.

Bonuses (Appendix B) and penalties (Appendix C) are additional tools for evaluation of the problem. An example of a bonus is a placement of the black King on the centre of the board. On the other side, if the black King is on the edge or in the corner, the position gets a penalty. The change in checks, which Black can give before and after the keymove, is another example of bonuses and penalties.

2.3 Complexity of CHESS COMPOSER

Chess composition rules do not allow a chess problem to contain more than one King, one Queen, two Rooks, two Bishops, two Knights, and eight Pawns for each colour (Harley, 1970; Howard, 1961). In this model, we keep to these rules. However, in order to estimate the complexity, we assume, in contrast to chess composition rules, that each one of the *p* pieces can be added on each step on an empty square.

Let *p* be the number of pieces we can add, and let *s* be a number of empty squares. Assuming *d* to be the number of the additions in this additions-tree, the number of nodes on level *d* by naïve additions is:

$$ps * p(s-1) * p(s-2) * ... * p(s-d+1) = p^d * \frac{s!}{(s-d)!} \qquad (1)$$

For instance, the 1-level enables *p***s* additions for each kind of piece on each empty square. The 2-level enables *p**(*s*–1) additions, because the number of the empty squares was reduced by one. Obviously, many positions are repeated by permutations of additions. For example, a White Bishop can be added before a Black Knight and visa-versa.

To prevent these repetitions, we apply an order on the added pieces as follows: Q (white Queen) > R (white Rook) > B (white Bishop) > N (white Knight) > P (white Pawn) > q (black Queen) > r (black Rook) > b (black Bishop) > n (black Knight) > p (black Pawn), which means that if we have added a piece from the list (say N), then we cannot add a piece which is preceding in the list (Q, R, B).

Applying this order we get the following formula for the number of nodes on level *d* of the additions-tree without repetition, assuming *d* to be the number of the additions:

$$\frac{s!}{(s-d)!} \sum_{i_{d-1}=1}^{i_d=p} \sum_{i_{d-2}=1}^{i_{d-1}} \sum_{i_{d-3}=1}^{i_{d-2}} \cdots\cdots \sum_{i_2=1}^{i_3} \sum_{i_1=1}^{i_2} \sum_{i_0=1}^{i_1} 1 \qquad (2)$$

Figure 2 presents the tree until depth *d* = 2. The number of possible additions is 10 (for [Q … p]) + 9 (for [R … p]) + 8 (for [B … p]) + . . . 1(p) = 55, exactly as expressed in formula (2). The implementation of the order on additions is quite simple. The equality (3) can be proved by induction on *d* (the number of the additions).

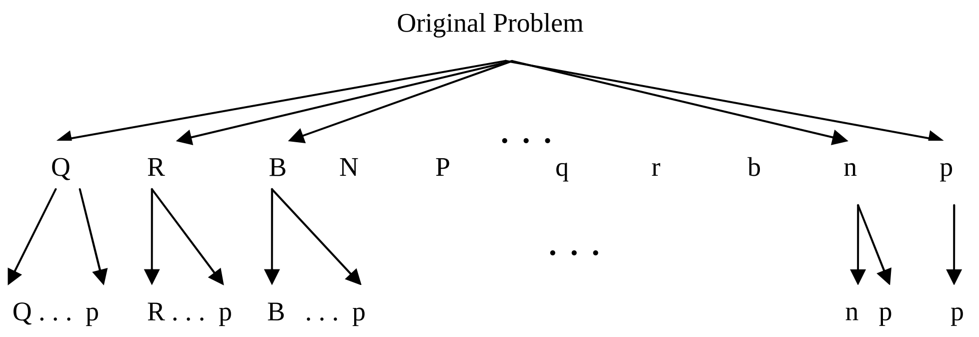

Figure 2: The additions-tree up to depth d = 2.

$$(2) = \frac{s!\prod_{1}^{d}(p+i-1)}{(s-d)!d!} \qquad (3)$$

Dividing (3) by (1) shows a great asymptotical improvement compared with formula (1). That is, our special order of additions improves the order of our generation algorithm.

$$\frac{(3)}{(1)} = \frac{\prod_{i=1}^{d}(p+i-1)}{p^d d!} = \frac{\prod_{i=1}^{d}(p+i-1)}{\prod_{i=1}^{d} pi} = \prod_{i=1}^{d}\frac{p+i-1}{pi} = \prod_{i=1}^{d}\left(\frac{1}{i}+\frac{1}{p}-\frac{1}{pi}\right) \xrightarrow[d\to\infty]{p>1} \frac{1}{p^{\infty}} \to 0 \qquad (4)$$

Taking into consideration also the two other kinds of transformations: deletions and transparencies, we can achieve the complete order: (1) deletions are done first, then transparencies, and eventually additions, (2) in case of addition of the same pieces, the order is due to the nearness to 0-square (h8 = 0; a8 = 7; h1 = 56; a1 = 63), and (3) order on deleted pieces regarding their nearness to 0 (i.e., to h8). After applying all these, we obtain a complete order on transformations, and the algorithm becomes d! faster compared with the naïve algorithm (formula 1) in cost of applying the order on transformations.

Deletions become relatively important when we treat positions with many pieces. This is because of the already mentioned chess composition rule about number of pieces: there cannot be more than 8 Pawns, 2 Bishops, 2 Rooks, 2 Knights, and 1 Queen for each colour and two same coloured Bishops cannot be placed on squares with the same colour.

Our theory is supported by experimental results. Each chain containing the same d transformations leads to the same position. That is, the same position can be reached in d! different permutations. Thus, actually, on the d-level we need to explore only $O(b^d/d!)$ nodes. The results presented in Table 1 support this theory. For instance, for d=1, the number of generated positions is 368 (= $368^1/1!$) on average; for d=2, the number of generated positions is 68,486 on average, compared with $368^2/2!$ (= 67,712); for d=3, the number of generated positions is 8,014,304 on average, compared with $368^3/3!$ (= 8,306,005). The running-time, needed for investigating all generated nodes on the first 3 levels for an average problem, is approximately 32 minutes.

Although our model implements a soft real-time system rather than a hard real-time, the run-time is still important. Therefore, we speed the model up by using a 64-bit representation (details in Appendix E).

	# of generated positions for all 100 problems			
	Average # of nodes	**Deviation (nodes)**	**Average time (seconds)**	**Deviation (seconds)**
1-level	368	83	0.16	0.07
2-level	68,468	28,156	20.84	7.14
3-level	8,014,304	4,522,082	1,911.39	775.69
4-level (for only 6 problems)	842,936,433	289,034,621	168,272.00	57,785.00

Table 1: Average number of generated positions by CHESS COMPOSER and times for all 100 original problems.

2.4 64-bit Representation

Adelson-Velskiy *et al.* (1970) were the first to propose an approach of 64 bits for board representation in the chess program KAISSA. Other programmers used the same approach with success in their chess-playing programs (Slate and Atkin, 1977; Heinz, 1997; Hyatt, 1999). The main ideas behind this method are economy of loops, absence of checking whether a piece moves out of the board, and the ability to compute complex calculations just in a few bitwise operations (Adelson-Velskiy *et al.* 1970, Slate and Atkin, 1977).

In this method, each kind of pieces has a unique 64-bit machine word (see Appendix E). Adelson-Velskiy *et al.* (1970) used the M20, a Russian computer. According to them, the length of the word in this machine was less than 64 bits. State and Atkin (1977) used CDC 6400, which also did not have 64 bits. The use of 64 bits in nowadays computers becomes increasingly wider. Heinz's DARKTHOUGHT and Hyatt's CRAFTY use 64-bit machines.

In our experiments, we use two 64-bit machines: AMD64 and Itanium-2. The output of our move generator is an array of new positions. All positions in the array are legal, assuming that the previous position is legal too (see details of our implementation in Appendix E).

Our move generator is fast enough compared to the known programs. It gives a speed of more than 22 million positions per second if we develop all variants starting from the initial positions using simple DFID. Castling and en-passant are allowed. Some known professional programs (e.g., CHESSEXPLORER at http://www.geocities.com/explorer127pl) that use the 32-bit approach perform analogously with a speed of about 15 million positions per second. Using the capability of 64 bits, the CHESS COMPOSER's (on AMD64 3400+) move generator is faster (see Table 2).

Pos./s.	**AMD64+ilogb**	**AMD64+bsf/bsr**	**Itanium2+ilogb**
'gcc –m64'	18,401,251	**22,163,976**	4,873,584
'gcc –m32'	7,109,440	7,445,146	**-**

Table 2: Generation of all possible positions from the initial chess position up to 8 plies on AMD64 with different configurations and on Itanium2.

Table 2 presents different runs of our move generator on different platforms with different configurations. To estimate speed, the initial chess position is developed up to depth 8. The speed is defined as number of generated positions developed per run-time of the program. We use AMD64-3400+ running on Linux 2.6.11 and compile with GCC 3.4.3. Functions λ and μ, respectively, stay on the least and the most important bit that is on, following Adelson-Velskiy *et al.* (1970). 'ilogb' and 'bsf/bsr' are 2 possible implementations for λ and μ (details in Appendix E). '-m64' and '-m32' are two different compilation parameters for gcc, complying into the 64-bit platform and into the 32-bit platform.

The main difference between generators based on rotated bitboards and our generator is the absence of use in rotated bits in our generator. The reason for this is the need to manage 3 additional bitboards with occupied pieces. "Looping" through different directions of sliding pieces is efficient enough, especially on a 64-bit architecture. However, we have a use in pre-calculated ranks for horizontal moves of Rooks and Queens (see "rm" in Table 8). It must be also pointed out that our move generator is not fully optimized. Therefore, its results are more impressive in comparison with 32-bit oriented programs (32% speed-up compared to CHESSEXPLORER mentioned above*).*

2.5 Comparison to Other Models

CHESS COMPOSER, in contrast to ICP, uses only three kinds of transformations. Therefore: (1) the branching factor of CHESS COMPOSER is smaller, and (2) the depth of the tree (the number of the applied transformations) developed by CHESS COMPOSER should be higher.

ICP uses a hill-climbing search, based on a heuristic function, to find local maxima in the near problem space. Hill-climbing search is an informed search. That is, it uses information of the newly developed nodes (Russell and Norvig, 2002) in order to prune some of the sub-trees.

In contrast, CHESS COMPOSER uses a brute-force search. This is an uninformed search. That is, it does not depend on any information contained in the newly developed nodes. This approach overcomes the main weakness of ICP, which is not developing the sub-trees whose roots are:

(1) not legal according to the rules of chess composition;
(2) not a two-move mate problem with only one key move;
(3) of a lower quality score than the original problem.

CHESS COMPOSER develops all possible transformations at all levels (see Table 1 for numbers). That is, it traverses all nodes to a fixed depth in order to find the best improvement (global maximum).

The main differences between CHESS COMPOSER and other previous models related to computer composition of chess-like and chess mate problems (mentioned in Section 1) are:

(1) CHESS COMPOSER uses a move-forward technique from the given problem instead of using either move-backward or random techniques;
(2) CHESS COMPOSER tries all possible transformations for every examined position, not only forced legal chess moves; therefore, its branching factor is much higher;
(3) CHESS COMPOSER starts with 2-movers and ends with them; there is no change in the number of the moves, in contrast to the model presented by Schlosser (1988, 1991);
(4) CHESS COMPOSER deals with a relatively high number of composition themes which are treated automatically by a computer rather than a human.

3 FOUR ILLUSTRATIVE EXAMPLES

In this section, we present four illustrative examples of problems improved by CHESS COMPOSER. Only one of the problems has been improved by ICP, but a much better improvement was found by CHESS COMPOSER. The three other problems have not been improved by ICP.

3.1 Example 1

The example starts at the miniature presented in Diagram 1. The composition theme expressed in this problem is “self-blocking”. The solution to is the key move **1. Kc7-d8**. There are 13 possible variants, only 4 of them express “self-blocking”. All 9 other variants include the same mate-move and do not contribute any novelty.

ICP, trying to improve the problem, applies the following rank-transparency transformation “moving all pieces 2 ranks below”. As a result, we reach the problem of Diagram 2. The solution is the key move **1. Kc5-d6.**

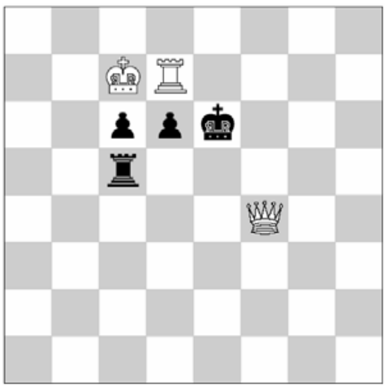
Diagram 1: Werner Speckmann, *Neueste Kieler Nachrichten*, 1939.

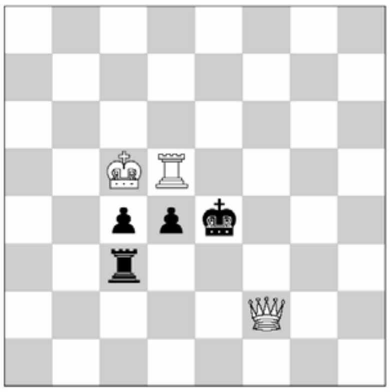
Diagram 2: The best improvement by ICP.

ICP’s problem is considered better than the original problem for the two reasons.

(a) The Black King in Diagram 2 is placed on a central square. For White, this is considered harder to mate.
(b) The problem of Diagram 2 includes 11 possible variants (versus 13 in the original), 4 of them (as in the original problem) express “self-blocking”. Only 7 other variants (versus 9 in the original problem) do not contribute any novelty. The four thematic variants are:

Black's move	White's mate-move	Themes included in the variant
(a) 1... Pd4 - d3	2. Rd5 - d4	
	2. Rd5 - e5	self-blocking; (thematic minor dual)
(b) 1... Rc3 - f3	2. Qf2 - d4	self-blocking
(c) 1... Rc3 - e3	2. Qf2 - f5	self-blocking
(d) 1... Rc3 - d3	2. Rd5 - e5	self-blocking

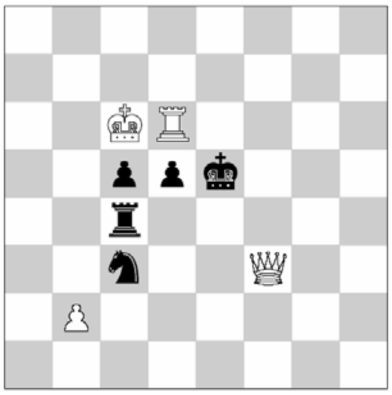
Diagram 3: The best improvement by CHESS COMPOSER.

CHESS COMPOSER applies the following three transformations: "rank-transparency transformation by moving all pieces one rank below", "addition of a white Bishop on b2", and "addition of a black Knight on c3". As a result, we reach the problem of Diagram 3. The solution for this problem is the same as for the problem of Diagram 1: the key move **1. Kc6 - d7**. There are nine possible variants. Only four of them are thematic (i.e., express "self-blocking"), as follows.

Black's move	White's mate-move	Themes included in the variant
(a) 1... Pd5-d4	2. Rd6 – e6	self-blocking
(b) 1... Rc4- f4	2. Qf3 – d5	self-blocking
(c) 1... Rc4- e4	2. Qf3 – f6	self-blocking
(d) 1... Rc4- d4	2. Rd6 – e6	self-blocking

The problem of Diagram 3 is considered much better than the original of Diagram 1 and than the ICP's improvement of Diagram 2. The comparison between the original problem and the ICP's improvement has already been presented. The comparison between the problems of the Diagrams 2 and 3 is given below. The position of Diagram 3 has several advantages compared to the problem of Diagram 2. We mention 2 of them.

(1) There are only 5 non-thematic variants in the problem of Diagram 3, while there are 7 non-thematic variants in the problem of Diagram 2.
(2) One of the thematic variants in the problem of Diagram 2 has a serious disadvantage. It is a thematic minor dual (i.e., there are two possible mate-moves for White instead of one). In contrast, the problem of Diagram 3 contains only pure thematic variants.

The only disadvantage of the problem of Diagram 3 compared to the others is that the others include only 7 pieces (i.e., they are regarded as miniatures), while the problem of Diagram 3 is not a miniature.

3.2 Example 2

The miniature (a problem that contains at most 7 pieces), presented in Diagram 4, has not been improved by ICP. However, it has been improved significantly by CHESS COMPOSER to the problem on Diagram 5. The composition themes expressed in the problem on Diagram 4 are: (1) "Grimshaw", which means that two black pieces block each other's line by interfering on a same square causing different mate variations and (2) "Self-blocking", which means that a black piece blocks one of black King's flights and by that creates different mate variations (Appendix A).

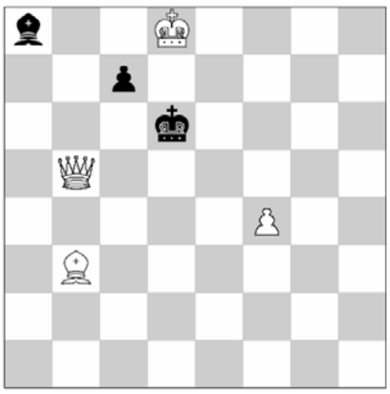
Diagram 4: H. Weenink, *Good Companion*, 12/1917.

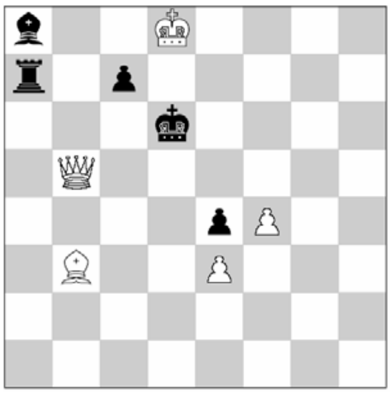
Diagram 5: The best improvement by CHESS COMPOSER.

The solution to the problem of Diagram 4 is the key move: **1. Qb5-c4**. There are 9 possible variants; only 3 of them are thematic (full details in Appendix D).

The best improvement to the problem of Diagram 4, found by CHESS COMPOSER, contains the following three transformations: “addition of a white Pawn on e3”, “addition of a black Pawn on e4”, and “addition of a black Rook on a7”. As a result, we reach position of Diagram 5.

The solution to the problem of Diagram 5 has the same key move: **1. Qb5-c4**. Then, there are 12 possible variants; 5 of them are thematic (see Appendix D).

The problem of Diagram 5 is considered to be better than the problem of Diagram 4 for the following three main reasons.
(1) The problem of Diagram 5 includes two pairs of the complex composition theme “Grimshaw” (i.e., 4 variants) instead of one pair (i.e., 2 variants) as in the problem of Diagram 4. The two Grimshaw squares are c6 (Pawn and Bishop) and b7 (Rook and Bishop). Problems that contain two pairs of the Grimshaw theme are relatively rare and hard to accomplish. Therefore, this addition is considered as a meaningful improvement of the original problem.
(2) The problem on Diagram 5 contains 5 thematic variants out of 12 (42%), while the problem of Diagram 4 contains only 3 thematic variants out of 9 (33%).
(3) Moreover, after the key move in the problem of Diagram 4, White has a mate *threat* 2. Qc4xc7. In the problem of Diagram 5 White has no mate threat; yet White succeeds in mating Black in two moves. This feature is called “*tempo*”, and it is also considered as an additional composition theme. Such problems are harder for humans to solve (Harley 1970).

The added Pawns in Diagram 5 are necessary for the solution. The black Pawn on e4 prevents Black to make moves by their Bishop on e4, f3, g2, and h1, moves that are not needed. The white Pawn on e3 prevents the black Pawn from moving.

There is only one slight disadvantage in the problem of Diagram 5 compared to the problem of Diagram 4: Diagram 4 includes only 7 pieces (i.e., it is a miniature), while Diagram 5 contains 10 pieces. The full evaluation of the problem of Diagram 4 and the problem of Diagram 5 is given in Appendix D.

3.3 Example 3

The problem presented in Diagram 6, has not been improved by ICP. It has been improved significantly by CHESS COMPOSER viz. to the problem of Diagram 7.

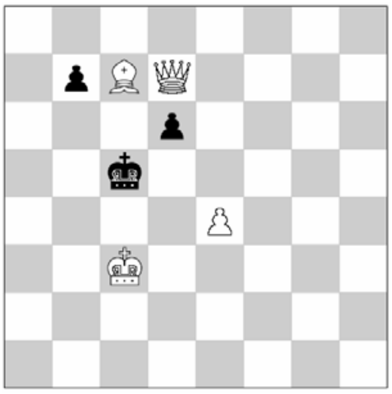

Diagram 6: A. Pasella en dr. G. Christofanini *Revista de Sah*, 1926. Also in *Jaarboek 1980*, Pr. # 13, page 29.

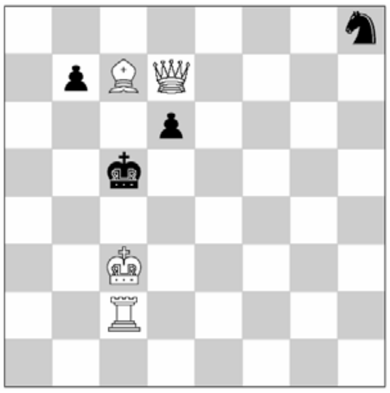

Diagram 7: The best improvement by CHESS COMPOSER.

The solution of the problem of Diagram 6 is the key move **1. Bc7-a5**. The variants are:

Black's move	**White's mate-move**	**Themes included in the variant**
(a) 1... b7-b6	2. Ba5-b4	self-blocking
(b) 1... d6-d5	2. Qd7xd5	
(c) 1... b7-b5	2. Qd7-c8	self-blocking
	2. Qd7-c7	self-blocking; (thematic major dual)

The solution of the problem of Diagram 7 is the key move **1. Qd7-e6**, threatening Qe6-c4 ±. The thematic variants are as follows.

Black's move	**White's mate-move**	**Themes included in the variant**
(a) 1... d6-d5	2. Qe6-b6	self-blocking
(b) 1... b7-b5	2. Qe6-d6	self-blocking
(c) 1... Kc5-c6	2. Kc3-b4	self-pinning, direct battery

The improved problem is much better because it does not contain any thematic major dual variant. A minor dual is a dual, of which all variants are forced in other lines of play. A major dual is a dual which is not a minor dual (Harley, 1970). The problem of Diagram 6 has a major dual because both mate-moves, Qc8 and Qc7, are not forced in other variants. Major duals are a serious flaw. Major duals in thematic variants are even worse.

The new problem also includes two new themes: self-pinning and direct battery. However, the self-pining is rather pseudo thematic: the pinned Pawn is not the reason to the mate-move. By self-pinning we mean a situation where Black moves and pins a piece and, as a cause of the pinning, Black is mated because the pinned piece cannot make a defensive move.

The self-blocking variant is expressed in two pure variants instead in one. In general, there are three pure unique variants (not counting the threat variants) instead of two. In the new problem, there is a mate move by the King, which is regarded as creative. Finally, after the key move, the black King has two King-flights (c6 and b5) instead of one (c6). This situation is regarded as the King-flight theme and it was absent in the original problem.

On the first look, the Knight on h8 does not contribute anything. It is an additional piece that does not add any thematic variant: variants which end with the threat mate are usually regarded as non-thematic. However, if the black Knight is not added, the problem is cooked.

The disadvantages are as follows: (1) the new problem is no more a miniature, and (2) it is no more a tempo problem.

3.4 Example 4

The problem presented in Diagram 8, has not been improved by ICP. It has been improved significantly by CHESS COMPOSER to the problem of Diagram 9.

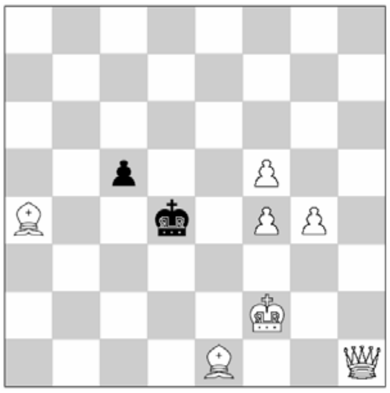
Diagram 8: *Israel chess magazine*, Pr # 1953.

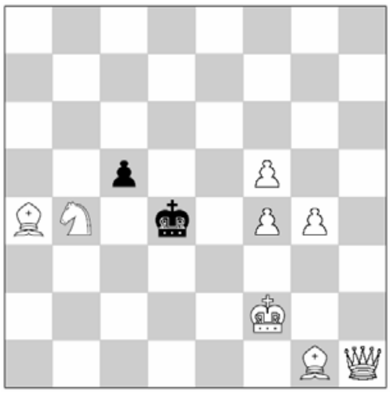
Diagram 9: The best improvement by CHESS COMPOSER.

The solution of the problem of Diagram 8 is the key move **1. Ba4-c6**, threatening 2. Qh1-d5 and 2. Qh1-e4. The variants are as follows.

Black's move	**White's mate-move**	**Themes included in the variant**
(a) 1... c5-c4	2. Qh1-d5	
(b) 1... Kd4-c4	2. Qh1-d5	
(c) 1... Kd4-d3	2. Qh1-e4	

The solution of the problem of Diagram 8 is the key move **1. Nb4-a2**. The variants are as follows.

Black's move	**White's mate-move**	**Themes included in the variant**
(a) 1... c5-c4	2. Kf2-e2.	self-blocking, direct battery
(b) 1... Kd4-c4	2. Qh1-e4.	
(c) 1... Kd4-d3	2. Qh1-d5.	

The improved problem includes two new themes: self-blocking and direct battery. All variants are uniquely different, while in the original problem the first and the second variants represent the so-called black dual. The new problem is also a tempo problem, which is better than the double threat in the original problem. If we look at the key moves, Ba4-c6 is a move towards the black King, while Nb4-a2 is the move in the opposite direction. The last feature is also regarded as an improvement.

4. RESULTS AND DISCUSSION

CHESS COMPOSER has been tested on 100 chosen 2-movers collected from different chess composition books. Most of the problems were collected from relevant composition books and correspondences (Haymann, 1988-1991; Howard, 1961, 1962, 1970; Harley, 1970; Thulin, 2000, 2003, 2004; Török, 2001, 2002; Godbout, 2001). Thirty six of them were the test set for ICP.

Figure 3 shows that new transformed positions are usually not legal by chess rules (about 30%) or not legal by chess composition rules ("not 2-movers" – about 63%). The problems are legal in about 7% of cases, but most of them are of a worse quality compared to the original problem. Only 0.32% of the cases are problems with a better score.

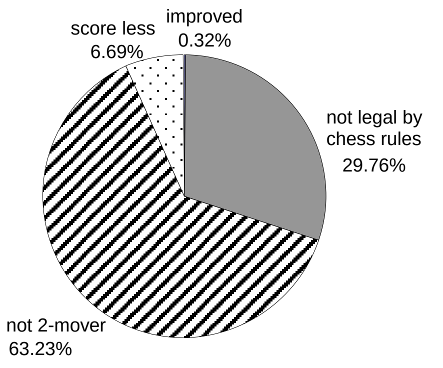

Figure 3: Distribution of transformed new positions for all levels.

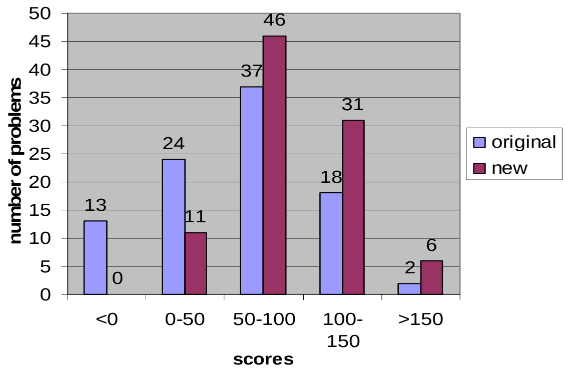

Figure 4: Change in scores.

Figure 4 presents the total change in scores from the original problems to the best scored problems. While 37 original problems have a score less than 50, only 11 improved problems have such a score. While only 57 original problems have a score greater than 50, there are 87 improved problems that have such a score.

Tables 1 and 3 show that the more nodes are developed, the more improvements are found. Our goal is to find the best improvements (if they exist). The chance to find a better improvement grows with the number of generated nodes. However, the growth is not linear (see percentages in Table 3).

It must be pointed out that the quality of a new problem does not depend linearly on a number of applied transformations. The problem mentioned in Diagram 4 has changed to the improved problem presented in Diagram 5 by using three transformations. If we use only one or two transformations from this set of three, the positions are even not legal! That is why the use of a hill-climbing search by ICP failed. ICP cannot deal with newly generated problems which are even not 2-movers.

# of problems	**# of improvements**			**Total # of improvements**
	1-level	**2-level**	**3-level**	
Average # of nodes	5	371	24,356	24,732
Deviation (nodes)	7	554	39,803	40,332
	improvements/nodes per level			
improved/generated	1.52%	0.56%	0.32%	

Table 3: Number of all generated improvements by CHESS COMPOSER for all 100 original problems.

On average, each problem was enriched by additions either in thematic variants (see Figure 5) or by an additional theme (see Figure 6). Almost half of the problems (48) faced a significant improvement because additional themes were added (see Figure 6). Two problems faced a reduction in one theme. The explanation is that in these problems there are thematic duals and triples, which are regarded as penalties. The CHESS COMPOSER's evaluation function found new problems with no duals at the cost of a decrease in themes.

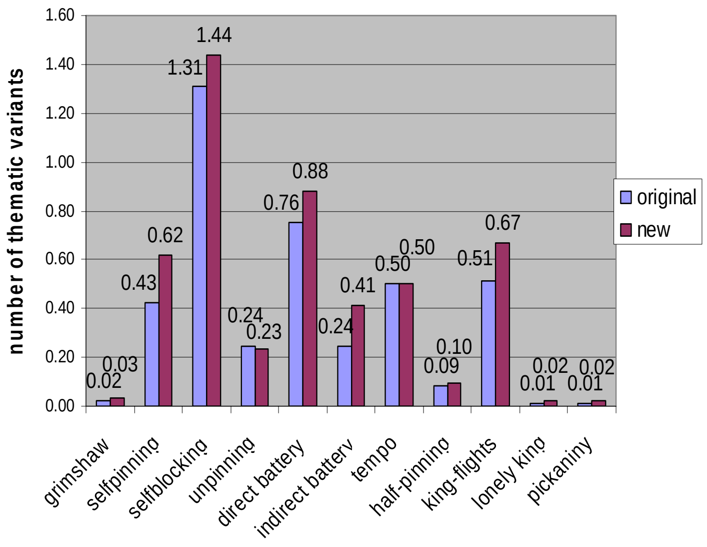

Figure 5: Average growing in themes.

Figure 7 shows that there is an increase in the number of pieces at 87 improved problems. Generally, due to the minimalism principle, the “dressing of the board” reduces the quality of the problem in modern chess composition (Harley, 1970). However, despite this fact, many meaningful improvements have been found.

As shown in Table 3, a sequence of three transformations is the most frequent case. The same is right in the case of the best scored improvements (78 cases in Figure 8). However, in 41 problems just one piece was added (Figure 7). The explanation is as follows: Figure 8 shows that 45 improvements contain at least the following two transformations: “deletion of a piece” and “addition of a piece”. This sequence can represent either a “moving of a piece” or a “change of a piece to another piece”, or a combination of the change and the moving (see the transformations applied in HaCohen-Kerner *et al.* (1999)). Another finding, shown in Figure 8, is that an addition of a piece is the most contributing transformation.

In Figure 9, we can see that, as expected, the time is decreasing as function of the number of pieces in the initial position. This is because the more pieces are on the board, the less free squares are left. Therefore, the “addition” transformation gives fewer possibilities. The decreasing line is not straight because of the other component: the already mentioned rule of chess composition. There can be no more than 1 Queen, 2 Rooks, 2 Bishops, 2 Knights, and 8 Pawns for each colour. Thus, as well as there are more pieces on the board, time needed to develop all positions becomes more problem-depended.

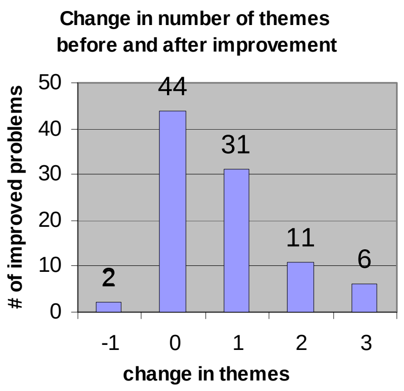

Figure 6: Almost half of the problems was enriched by additional themes.

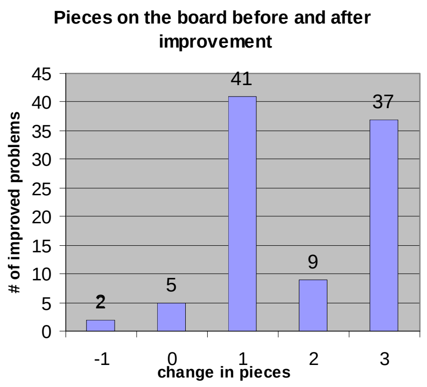

Figure 7: Increase of the pieces in best-scored problems.

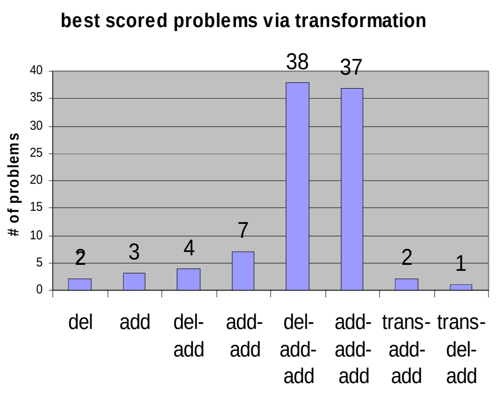

Figure 8: Transformations distribution.

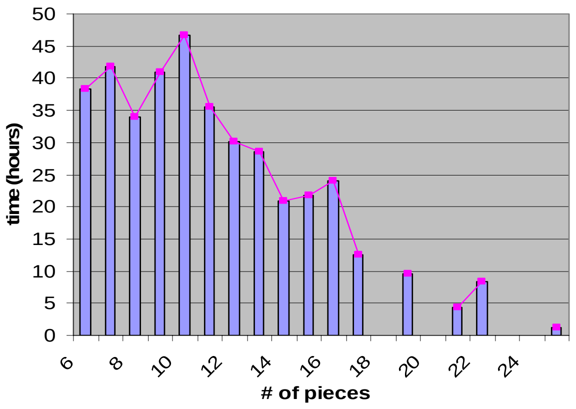

Figure 9: Run-time as a function of the number of pieces.

There are three problems that Chess Composer failed to improve, even after four transformations. Three other problems have been improved slightly only after four transformations. These 4-transformation improvements did not add themes. All these six problems have several common features: (1) they include various themes, (2) there are neither duals nor triples nor multiples, and (3) every (or almost every) variant is thematic. In other words, these problems are perfect (or almost perfect).

5. SUMMARY AND FUTURE WORK

Chess Composer presents an ability of improving the quality of 97 of 100 known chess problems, composed by human composers. Most of the improvements were achieved after various sequences of three transformations (mainly three additions or a deletion with two additions), while the most contributing transformation was “an addition of a piece”. The successful sequences of transformations improved the quality of the problems by increasing the average number of themes, the average number of unique thematic variants, and the average proportional rate of the thematic variants.

Some of the improvements are rather impressive, considering that most of the tested problems were composed by experienced composers. These new improved problems can be regarded as creative from the viewpoint of experts in chess composition because these problems are better and they are not too similar to the original problems.

The main disadvantage of Chess Composer is its relative slowness. Therefore, we apply sequences of 3 and sometimes 4 transformations only. This is because we attempt to investigate all possible positions using a brute-force search method, while working with a very large branching factor. A possible first direction of future research is smart pruning of non contributing sub-trees (estimated as such by a heuristic). It may enable searching to deeper levels. A second direction may be improving the quality function, it might contribute also to such a pruning.

A third possible research direction is the creation of problems from scratch. There are a few ways to generate random 2-movers. For instance, (1) searching for 2-movers in the databases generated by either Thompson (1986, 1996) or Nalimov *et al.* (2000), (2) in a similar way to Schlosser (1988, 1991) we can start from given mating positions, move backward 3 plies and check the possible positions for a creation of 2-movers. Then, Chess Composer can be applied to these 2-movers. It would be interesting to compare the quality of the problems generated this way to the quality of the problems generated from previous composed 2-movers.

Finally, we remarkt that the current model can be extended to *k*-move ($k > 2$) mate problems. This extension will demand: (1) smart pruning and/or use of faster search algorithms and (2) extending the quality function based on the definitions of various special new themes dedicated to *k*-move problems, bonuses, and penalties, as it follows from Howard (1970).

6. ACKNOWLEDGEMENTS

The authors would like to thank Josef Retter and Jean Haymann, both international masters of the FIDE for chess composition. They supplied us with relevant background information on the composition of chess problems. Thanks also to Yair Wiseman for helpful comments on the 64-bit representation and Olga Fainshtein for general comments. Finally, the three anonymous referees are gratefully recognized for their valuable comments.

7. REFERENCES

Adelson-Velskiy, G. M., Arlazarov, V. L., Bitman, A. R., Zhitvotovskiy, A. A., and Uskov, A. V. (1970). Programming a Computer to play Chess. *Russian Mathematical Survey*, Vol. 25, pp. 221–262.

Godbout, A. J. (2001). Echos du Nord, 256 Problèmes choisis de Johan Scheel. *Éditions de l'Apprenti Sorcier.* ISBN 0-9688828-6-2.

HaCohen-Kerner, Y., Cohen, N. and Shasha, E. (1999). An Improver of Chess Problems. *Cybernetics and Systems*, Vol. 30, No. 5, pp. 441–465.

Harley, B. (1970). *Mate in Two Moves*, Dover Publication, Inc., New York, NY.

Hartmann, D. (1989). Notions of evaluation functions tested against grandmaster games. *Advances in Computer Chess* 5 (ed. D.F. Beal), pp. 91–141. North Holland Elsevier, Amsterdam, The Netherlands. ISBN 0-444-87159-4.

Haworth, G. (2003). Reference Fallible Endgame Play. *ICGA Journal*, Vol. 26, No. 2, pp. 81–91. ISSN 1389-6911.

Haymann, J. (1988–1991). First Steps in Composition. A Series of Correspondences. *Variantim: Bulletin of the Israeli Chess Composition Association*.

Heinz, E.A. (1997). How DARKTHOUGHT Plays Chess. *ICCA Journal*, Vol. 20, No 3, pp. 166–176. ISSN 0920-234X.

Heinz, E.A. (1999). Endgame Databases and Efficient Index Schemes. *ICCA Journal*, Vol. 22, No. 1, pp. 22–32. ISSN 0920-234X.

Howard, K. S. (1961). *The Enjoyment of Chess Problems*. Dover Publication, Inc., New York, NY.

Howard, K. S. (1962). *One Hundred Years of the American Two-Move Chess Problem*. Dover Publication, Inc., New York, NY.

Howard, K. S. (1970). *Classic Chess Problems by Pioneer Composers*. Dover Publication, Inc., New York, NY.

Hyatt, R. M. (1999). Rotated Bitmaps, a New Twist on an Old Idea. *ICCA Journal*, Vol. 22, No. 4, pp. 213–222. ISSN 0920-234X.

Kerner (HaCohen-Kerner), Y. (1995). Learning Strategies for Explanation Patterns: Basic Game Patterns with Application to Chess. Case-Based Reasoning: Research and Development, *Proceedings of the First International Conference, ICCBR-95* (eds. M. Keane, J. Haton and M. Manago). Lecture Notes in Artificial Intelligence 1010, pp. 491–500, Springer-Verlag, Berlin. ISSN 0302-9743.

Korf, R. E. (1985). Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, Vol. 27, No. 1, pp. 97–109. ISSN 0004-3702.

Nalimov, E. V., Haworth, G. McC., and Heinz E.A. (2000). Space-Efficient Indexing of Chess Endgame Tables. *ICGA Journal*, Vol. 23, No. 3, pp. 148–162. ISSN 1389-6911.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2n^{d} Edition, Place. ISBN 0-13-790395-2.

Schlosser, M. (1988). Computers and Chess-Problem Composition, *ICCA Journal*, Vol. 11, No. 4, pp. 151–155. ISSN 0920-234X.

Schlosser, M. (1991). Can a Computer Compose Chess Problems? *Advances in Computer Chess* 6 (ed. D.F.Beal), pp. 117–131. Ellis Horwood Ltd., Chichester, UK. ISBN 013-0065.

Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine, Vol. 41, No. 7*, pp. 256–277. Reprinted (1988) in Computer Chess (ed. D.n.L. Levy), pp. 81-88, Springer-Verlag, New York, N.Y. ISBN 0387-96496-7.

Slate, D. J. and Atkin L. R. (1977). CHESS 4.5-The Northwestern University University Chess Program. *Chess Skill in Man and Machine* (ed. P.W. Frey), Springer-Verlag, New York, N.Y. 2nd ed., 1983, pp. 82–118. ISBN 0-387-90815-3. Reprinted (1988) in *Computer Chess Compendium* (ed. D.N.L. Levy), pp. 80–103. Batsford Ltd., Londen, UK. ISBN 0-7134-494-4.

Tamplin, J. and Haworth, G. (2001). Ken Thompson's 6-man Tables. *ICGA Journal*, Vol. 24, No. 2, pp. 83–85. ISSN 1389-6911.

Thompson, K. (1986). Retrograde analysis of certain endgames, *ICCA Journal*, Vol. 9, No. 3, pp.131–139. ISSN 0920-234X.

Thompson, K. (1996). 6-Piece Endgames. *ICCA Journal*, Vol. 19, No. 4, pp. 215–226. ISSN 0920-234X.

Thulin, A. (2000). John Thursby – Seventy-Five Chess Problems (1883), *electronic edition**.

Thulin, A. (2003). Frank Healey – 200 Chess Problems (1866), *electronic edition**.

Thulin, A. (2004). T.Taverner – Chess Problems Made Easy (1924), *electronic edition**.

Török, G. (2001). Karol Mlynka – 409 Selected Chess Problems. *Éditions de l'Apprenti Sorcier.* ISBN 0-9688828-7-0.

Török, G. (2002). Zoltan Labai – Selected Compositions. Volume 1 (1968–1982). *Éditions de l'Apprenti Sorcier*. ISBN 0-9688828-9-7.

Watanabe, H., Iida, H., Uiterwijk J. W. H. M. (2000). Automatic Composition of Shogi Mating Problems. *Games in AI Research* (eds. H.J. van den Herik and H. Iida), pp. 109–123, Institute for Knowledge and Agent Technology (IKAT), Universiteit Maastricht, The Netherlands. ISBN 90-621-6416-1.

*) downloaded from http://www.algonet.se/~ath/.

8. APPENDICES

Below we provide five appendices. They contain definitions (Appendix A), bonuses (Appendix B), penalties (Appendix C), evaluations (Appendix D), and 64-bit representation (Appendix E).

Appendix A: Definitions of composition themes and their relative value in points

# of theme	**Composition theme**	**Definition**	**Value in points**
1	Tempo or Waiting move	The key move doesn't threat any mate move.	10
2	Direct battery	A battery with a White piece in the middle of the battery where a battery is defined as follows: a piece is standing between a long-range piece (Queen, Rook or Bishop) and a King. *Prepared* battery is an existent battery after the key move. Battery *fires* when the attacking piece in the battery mates.	15
3	Indirect battery	A battery to a square adjacent to the King's square. It must be *prepared* and *fired* too. In addition, any other piece must not fire on the suspected as fired by indirect battery square.	25
4	King flights	The key move does not decrease the number of square/s that the black King can escape to (Howard, 1962).	15
5	Lonely king	Black has only a King.	2
6	Half-pinning	Two Black pieces standing between a white long-range piece (Queen, Rook or Bishop) and a black King. Each of the two black pieces must be a part of the self-blocking theme.	25
7	Self-pinning	The Black makes a move and pins one of his pieces. If there is a variant in which White mates and Black could prevent this mate by the pinned piece, than it is *self-pining* variant.	15
8	Unpin opponent piece	Black makes a move and unpins a white piece. The white piece must mate.	20
9	Self-blocking	A black piece blocks black's King. When the black King is mated the blocked square is not threatened by any white piece.	25
10	Grimshaw	Two black pieces which each one of them blocks the other's line by mutually interfering on the same square and by that causes different mate variations. The pieces are either sliding pieces or a jumping pawn.	45
11	Pickaniny	There are 4 variants for a black Pawn and each variant finishes with different mate. This happens only on the seventh horizontal.	25

Table 4: The values of composition themes.

Notes:

1) Only unique mates are counted.
2) For the first unique thematic variant the full number of scores is given; for the rest just the fifth.
3) Pin-related themes may be defined for opposite colour pieces (Harley, 1970; Howard, 1962). However, the most important themes were defined in the table consulting two international masters in chess composition.
4) Thematic variants do not contain threats after the key move.
5) *Tempo*, *King flights*, and *lonely King* are special themes without thematic variants.

Appendix B: Bonuses

# of bonus	Feature	Bonus in points
1	Miniature	10
2	Meredith	5
3	black King is in the centre	10
4	X pieces on board	3*(18-X)
5	Key by King	15
6	Mate move by King	20 for the first unique move, 4 for the rest
7	Key gives X more King-flights to Black	15*X
8	Key enables Black to check White X more times	30 for the first unique check, 6 for the rest (X-1) checks
9	Key pins a white piece	3 * piece's value
10	Key unpins a black piece	5 * piece's value
11	Key sacrifices a white piece	5 * piece's value
12	$X_1, \ldots, X_{11}$ variation occurrences for themes # 1, ..., 11, respectively	First theme's occurrence gives the full value in points of the theme. Each additional occurrence of the same theme in other variations adds only 1/5* full value. An exception is the theme "King-flights". Each variation of it adds the full value of this theme.
13	Mates' value	$\frac{10*(\text{\# of unique thematic mate moves})^2 + 3*(\text{rest unique mates})}{\text{\# of variants}}$
14	Distance of the keymove	The distance is defined as a distance between start and end placement of the keymoved piece, as like it has been done by a queen (possibly in two moves). The formula is 3*(diagonal part)+2*(horizontal or vertical part).
15	Distance to/from the black King	The difference in distances, in terms of 14, between the black King's square and start and end squares of the keymove (if the result is positive).

Table 5: The bonus of a feature.

Appendix C: Penalties

# of penalty	Feature	Penalty in points
1	X minor duals	X
2	X minor triples	2 * X
3	X minor multiples	3 * X
4	black King is in the corner	20
5	black King is in the edge	10
6	Key is a check	50
7	Key is a double check	70
8	Key is a capture of a black piece	10 * piece's value
9	Key is a promotion of a Pawn (unless it is the theme)	5 * piece's value
10	Key takes X King-flights from Black	15 * X
11	Key pins a black piece	5 * piece's value
12	Key unpins a white piece	3 * piece's value
13	If Value of white pieces > Value of black pieces	2 * (Value of white pieces -Value of black pieces). Values of pieces are according to Shannon (1950).
14	Distance to/from the black King	The difference in distances, in terms of 14 in bonuses, between the black King's square and start and end squares of the keymove (if the result is negative).
15	Thematic non-single	30

Table 6: The penalty of a feature.

Appendix D: The result of the improvement process from Diagram 4 to Diagram 5

Problem of Diagram 4

White: Kd8 Qb5 Bb3 Pf4   **Black**: kd6 ba8 pc7
The **key move** is: Qb5-c4

ORIGINAL POSITION EVALUATION:

```
THEMES:
    threat:             Qc4:c7
    selfblocks:         3      score:  35 (award is 25 for the first variant, 5 for the rest)
    grimshaw pairs:     1      score:  45 (award is 45 for each occurrence)
    ----------------------------------
               Total score for themes   :---> 80
BONUSES:
   position size:   miniature        score:  10
   pieces on the board:     7        score:  33 (award is due to 3*(18 - #pieces))
   *** different thematic mates: 3, different nonthematic mates: 2, variants:9
   score: 10 (price is due to (10*#diff thematic mates^2 + 3*other diff mates) / #variants)
   key-move distance:       3 (award is due to horizontals & verticals 2, diagonals 3 )
   ----------------------------------
               Total score for bonuses  :---> 56
PENALTIES:
   pieces value, white:13, black: 4  score:  18 (award is due to 2*(whites - blacks))
   ----------------------------------
               Total score for penalties:---> 18

POSITION SCORE ........................:--->118

VARIANTS:
1)   c7-c6        Qc4-d4 ±      self-blocking; grimshaw
2)   c7-c5        Qc4-e6 ±      self-blocking
3)   ba8-c6       Qc4-b4 ±      self-blocking; grimshaw
4)   ba8-d5       Qc4xd5 ±
5-9) ba8-any else Qc4xc7 ±
```

Problem of Diagram 5

White: Kd8 Qb5 Bb3 Pf4 Pe3   **Black**: kd6 ra7 ba8 pc7 pe4
The **key move** is: Qb5-c4

IMPROVED POSITION EVALUATION:

```
THEMES:
    tempo (waiting)           score:  10 (award is 10)
    selfblocks:        3      score:  35 (award is 25 for the first variant, 5 for the rest)
    grimshaw pairs:    2      score:  90 (award is 45 for each occurrence)
    ----------------------------------
                    Total score for themes   :--->135
BONUSES:
   position size:   meredith         score:   5
   pieces on the board:    10        score:  24 (award is due to 3*(18 - #pieces))
   *** different thematic mates: 4, different nonthematic mates: 1, variants:12
   score:  13 (award is due to (10*#diff thematic mates^2 + 3*other diff mates) / #variants)
    key-move distance: 3 (award is due to horizontals & verticals 2, diagonals 3 )
    ----------------------------------
                    Total score for bonuses  :---> 45
PENALTIES:
    pieces value, White:14, Black:10  score:  8 (award is due to 2 * (whites - blacks))
    ----------------------------------
                    Total score for penalties:--->  8
POSITION SCORE ............................:--->172

VARIANTS:
1)    c7-c6        Qc4-d4 ±     self-blocking; grimshaw
2)    c7-c5        Qc4-e6 ±     self-blocking
3)    ba8-b7       Qc4xc7 ±     grimshaw
4)    ba8-c6       Qc4-b4 ±     self-blocking; grimshaw
5)    ba8-d5       Qc4xd5 ±
6)    ra7-b7       Qc4-d5 ±     grimshaw
7-12) ra7-any else Qc4xc7 ±
```

IMPROVEMENT by addition of P on e3, addition of p on e4, addition of r on a7.
Score: 118 -> 172, 45%

Appendix E: 64-bit representation

Let us use the common denotations for the variables that represent the chess board in Table 7 and 8. Table 7 presents the primitives that are sufficient to store the whole position. In our program, the relationship between the board squares and its bits is as follows: a1, ..., h1, a2, ..., a7, ..., a8, ..., h8 where a1 is msb (bit number 63) and h8 is lsb (bit number 0).

WP – White Pawns	BP – Black Pawns
WN – White kNights	BK – Black kNights
WR – White Rooks	BK – Black Rooks
WB – White Bishops	BB – Black Bishops
WQ – White Queens	BQ – Black Queens
WK – White King	BK – Black King

Table 7: Primitives for board representation.

AW = WP | WN | WR | WB | WQ | WK - all white pieces	
AB = BP | BN | BR | BB | BQ | BK - all black pieces	
AP = AW | AB – all pieces	
NAP = ~AP – empty squares	
1 – first horizontal	*a* – vertical a
8 – eight horizontal	*h* – vertical h
~1 – all but first horizontal	~*a* – all but vertical a
~8 – all but eight horizontal	~*h* – all but vertical h

Table 8: Helpful variables.

Using bitwise operations "|" (OR) and "~" (NOT), we can immediately get 4 helpful 64-bit variables. These are AW, AB, AP, and NAP (see Table 8). The rest helpful variables in Table 8 are pre-calculated 64-bit words. "1" and "8" mean a 64-bit word in which bits are on the first and the eights horizontal respectively; bits are off on other horizontals. *a* and *h* are the same for verticals. By "~" before a number or before a letter *n* we mean another 64-bit word which is bitwise operation "not" on the *n* (also pre-calculated). Using primitives from Table 7 and helpful variables from Table 8, in addition to other bitwise operations "<<" (shift left), ">>" (shift right), and "&" (AND), we can easily get all Pawns moves and captures (see Table 9). For example, all Pawns' captures to the left, *clPW*, we get by ANDing all white Pawns, WP, with the negotiation of the vertical *a*, ~*a*, shifting the result one vertical left and ANDing again with all pieces of black (see line 5 in the Table 9).

Pawns	White	Black
Pawns' moves	mPW = (WP >> 8) & NAP	mPB = (BP << 8) & NAP
Pawns moves, no promotion	mPW~8 = mPW & ~8	mPB~1 = mPB & ~1
Pawns jumps	jPW = (mPW >> 8) & NAP	jPB = (mPB << 8) & NAP
Pawns captures to the left	clPW = ((WP & ~*a*) >> 7) & AB	clPB = ((BP & ~*a*) << 9) & AW
Pawns captures to the right	crPW = ((WP & ~*h*) >> 9) & AB	crPB = ((BP & ~*h*) << 7) & AW
Pawns captures to the left, no promotion	clPW~8 = clPW & ~8	clPB~8 = clPB & ~1
Pawns captures to the right, no promotion	crPW~8 = crPW & ~8	crPB~8 = crPB & ~1
Pawns moves, with promotion	mPW+8 = mPW & 8	mPB+1 = mPB & 1
Pawns captures to the left with promotion	clPW+8 = clPW & 8	clPB+1 = clPB & 1
Pawns captures to the right with promotion	crPW+8 = crPW & 8	crPB+1 = crPB & 1
Enpassant	Special case	Special case

Table 9: Pawn moves and captures.

We distinguish between "captures" and "regular" moves. This is because when a piece "captures", it influence the opposite side too. The promotion to the eight horizontal (or the first for Black) is also a special case. This is because 4 new positions (a Pawn can be promoted to 4 other kinds of pieces) are created instead of one in regular move.

A different approach is used for Knights and Kings (see Table 10). To obtain all legal moves and captures we use masks, *maskN* for Knights and *maskK* for Kings. The colour in this case makes no difference. Masks are arrays of size 64 of 64-bit words. Each mask represents moves of a piece. A 64-bit word on index *n* of an array means all moves of the piece if it would be on the square *n*.

To be able to use masks, we use a λ (bitword) function that returns the number of the least significant bit that is on. One of the possible implementations of λ is using of look-up tables. However, on some platforms there is a faster way to calculate λ. For example, on Intel Pentiums and Amd64 processors there is a special instruction *bsf* which is faster than the use of look-up tables. The disadvantage of the look up tables is their memory space consuming. So, the whole table will not fit in the cache and thus the use of it can be potentially slow. On architectures without this special instruction the code shown in Table 11 can be used instead of look up tables.

All Knight moves	amWN = maskN[λ (WN)]	amBN = maskN[λ (BN)]
Knight moves	mWN = amWN & NAP	mBN = amBN & NAP
Knight captures	cWN = amWN & AB	cBN = amBN & AB
All King moves	amWK = maskK[λ (WK)]	amBK = maskK[λ (BK)]
King moves	mWK = amWK & NAP	mBK = amBK & NAP
King moves	cWK = amWK & AB	cBK = amBK & AB

Table 10: Use of masks for Knights and Kings.

The idea is simple: *lsb_bit* gives the nearest to the lsb bit which is on. The number which has just 1 bit on is a power of some number. So we just find the binary logarithm. 'ilogb' is a relatively fast function.

`typedef unsigned long long uINT64;` `typedef long long INT64;` `inline int λ(const uINT64 a){` `return (0==a)?64:ilogb(lsb_bit(a));` `}`	`typedef unsigned long long uINT64;` `typedef long long INT64;` `inline int μ(const uINT64 a){` `return (0==a)?64:ilogb(a);` `}`
Where `lsb_bit` is defined as:	`inline uINT64 lsb_bit(const uINT64 a){` `return a & -(INT64)a;` `}`

Table 11: Possible code for functions λ and μ**.**

Sliding pieces are carried out differently. The reason is the ability of existing of a piece on the way of sliding. So, in addition to pre-calculated masks in the sliding direction ("rays" in Table 12), we need to cut all squares behind the blocking piece. The solution for down and up directions in case of white rooks is shown on Table 12. μ **(**bitword) is the number of the most significant bit that is on. Similar to λ, there are architectures with the special instruction *bsr* for μ**.** For the architectures without it, the code shown on Table 11 can be used.

Rooks	For each white Rook
Masks	Dr = down_ray(λ(WR); ur = up_ray(λ (WR); rm = rankMoves[λ(WR)][(char)(AP >> number of rank)]
Moves down	mR = dr & ((lsb_bit(dr & AP) << 1) – 1)
Moves up	mR |= ur ^ up_ray[μ(ur & AP)]
Moves left & right	mR |= rm
just moves	jmR = mR & NAP
just captures	jcR = mR & AB

Table 12: Moves and captures by sliding pieces on example of white Rooks.

For the left and right directions, the placement of a Rook and all other pieces on the Rook's horizontal determine all moves of the Rook. So, we can pre-calculate all possible situations and to store them into 2-demantions array (rankMoves[64][256]) similar to Heinz (1997) and Hyatt (1999). In addition to left-right directions, they also pre-calculate all other three pairs of directions for all sliding pieces. In purpose to use these pre-calculations, Heinz and Hyatt use so-called rotated and flipped bitboards. The disadvantage of the method is managing of additional bitboards. We have no use of flipped and rotated bitboards. Black Rooks are carried out the same as white Rooks. We obtain Bishop moves and captures in the same way as for Rook, but using 4 diagonal rays instead of vertical and horizontal ones. Queen moves are obtained by the same procedure as for Bishops and Rooks together. Using the same technique as for Queens in addition to Knights, we determine whether a King is still under attack and thus, the position is illegal. In this way, we cancel illegal positions fast. Using the same method, we check the "cannot castle" issue for 5 squares for each colour: c1, d1, e1, f1, g1 for White and c8, d8, e8, f8, g8 for Black. In addition, we must remember if a castling King or a Rook moved in the past. As in the case of en-passant capture, the cost is an addition of a word to the position structure (see Table 7).