CO-ORDINATE SQUARES:
A SOLUTION TO MANY CHESS PAWN ENDGAMES

by

Kenneth Church

Submitted in Partial Fulfillment

of the Requirements for the

Degree of Bachelor of Science

at the

Massachusetts Institute of Technology

June, 1978

Signature of Author Signature redacted

Department of Electrical Engineering

Certified by Signature redacted

Thesis Supervisor

Accepted by Signature redacted

Charmain, Departmental Committee on Theses

Archives

Co-ordinate Squares: a Solution to Many Chess Pawn Endgames

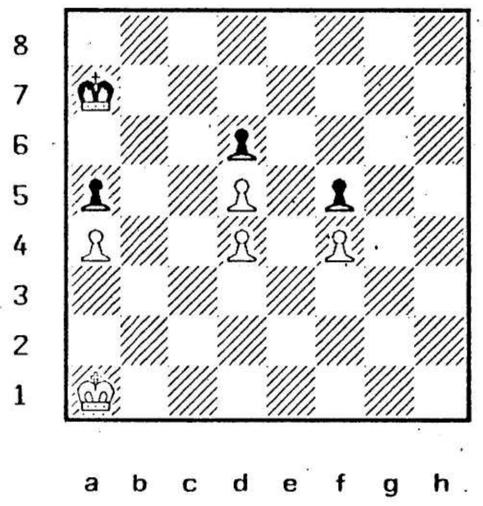
by Kenneth Church

adviser: Richard Greenblatt

Abstract:

The solution to a number of AI problems depends upon knowledge representation. The theory of *Co-ordinate Squares* is a framework for representing the relevant ideas in a large class of king and pawn endings. Some endgame problems which require twenty to thirty plies of analysis using a more traditional representation, a branching tree of moves, can be efficiently solved using this theory.

Figure 1



white to move wins; black to move draws
[Fine, problem #70] [Averbakh, problem #671] [Lasker 1901]

(The author of Peasant[1] estimates that this problem would require 25,000 hours of cpu time. [Frey, page 129])

Table of Contents

Table of Figures

What are Co-ordinate Squares?

A good representation of a chess endgame should be based upon as large an invariant as possible. That is, the concept of a move is too elementary and too transient to show what is going on. A tree of moves is consequently too large because the branches don't contain enough information. If the representation models something which doesn't change during the game or changes only very occasionally, then the structure will be much more manageable.

In this work, the representation is a map of *co-ordinate squares* [Averbakh, chapter 7]. This map shows us how one side, the defender, is co-ordinating with the other, the attacker. Intuitively, the defender is co-ordinating with the attacker if, for any move the attacker makes, the defender has a defending reply. Since we have limited ourselves to the restricted domain of king and pawn endgames, we can use domain specific constraints to determine whether the defender is co-ordinating with the attacker.

The most important observation is that there are two distinct classes of moves in king and pawns endgames, king moves and pawn moves. If we can de-couple these two classes, then there would be a relatively small manageable number of distinct positions that could result after an arbitrary series of moves. It is relatively easy to understand what will happen after an arbitrary series of reasonable pawn moves. Pawns can rarely queen without king assistance. Usually the pawns have to be passed or half-passed to be a threat. One method for locating the threatening pawns will be outlined in detail later in this paper. King moves are much more difficult since kings have a much larger branching factor than pawns. However, since there can be no more than 64^2 ways to place two kings on the board, the number of positions resulting after an arbitrary series of king moves is sufficiently constrained for our purposes. The final step, deciding how the king and pawn moves are coupled, is the most heuristic, requiring considerable domain specific knowledge. We will first study king moves, then pawn moves, and finally the coupling effect between the two classes of moves.

King Co-ordination

Let one king be the attacking king and the other be the defender. Can the defending king defend against all the attacking king's threats? If we can answer this question for all ways of placing the two kings, then we have the map of co-ordinate squares. Since the attacking king is free to move wherever he choses whereas the defending king is obligated to move to a defense of the attacking king, the map of co-ordinate squares is organized as sets of defenses for each *attacking square*, that is, for each placement of the attacking king.

What is a defense? A defense must satisfy two conditions: first it must be a possible defense according to a static evaluator. Secondly, it must be connected to other defenses so that there is a defensive reply to any single move by the attacker. The second condition is sufficiently strong to assure that the defending king can still defend after an arbitrary series of king moves by the attacker.

The fact that defenses are invariant is not completely obvious. One might imagine that, even though the defending king is holding against all threats, the attacker could still win by moving away and then returning at a more advantageous time. However, the connectedness condition is designed so that defenses will be invariant. Intuitively, if the attacker could out manoever the defender, then there must be a position where the attacker can make some move for which the the defender has no reply even though the defender was holding before the attacker moved. Since this contradicts the connectedness criterion, defenses must be invariant.

Given an attacking square, it is possible to determine the *defense set*, the set of all defenses to a given placement of the attacking king. We say that the defending king co-ordinates with the attacking king if, after the defender's move, the defending king occupies a square in the defense set of the attacking king. Since defenses are invariant, we can determine if co-ordination can ever be achieved by checking the defense set of the attacking king. If the location of the defending king is not in the defense set, then we know the defender is lost because he cannot achieve co-ordination. More generally, we can determine the result of any chess endgame (win, loss, or draw) if we know whether or not co-ordination can be achieved. There are only three possibilities: both sides are co-ordinating with each other, white is co-ordinating with black but not vice versa, or only black is co-ordinating with white. These positions are drawn, won for white, and won for black, respectively.

Only slightly more effort is necessary to find the best move. The king trying to win must maintain his winning position and avoid an infinite repetition. In terms of co-ordinate squares, maintaining a winning position simply means that the winning king must co-ordinate with the other king and prevent the other king from co-ordinating with him. The author has spent very little effort developing an attractive algorithm for avoiding repetition because co-ordinate squares so constrains the possibilities that almost anything will do. To draw, it is only necessary to avoid a lost position, a position where it is impossible to co-ordinate with the other king. Unfortunately, this theory doesn't help a lost position. One strategy might be that the losing king should play the move which gives the attacking king the greatest opportunity to make a mistake.

The difficulty with co-ordinate squares is that it is necessary to determine the defense sets in order to know whether or not co-ordination can be achieved. Unfortunately, this task can be extremely difficult and is most likely impossible in the general case. Much of this paper will concern itself with heuristics for generating these sets in specific cases.

The simplest case (such as figure 1 shown above) is where neither side has any pawn mobility and the defending king must defend all the opponent's threats (he cannot generate his own more immediate threats called *counter attacks*). Although this example is the first and most elementary to be considered, the reader is challenged to solve it. Even experts, USCF rating of 2000 to 2200, have been stumped. At any rate, for the purposes of explanation, let us consider black to be the defender. We are thus exploring the question of whether black is co-ordinating with white.

Finding Targets

First of all, we must determine the targets, which squares the attacking (white) king ultimately wants to achieve. Let us define a target for a particular side *s* to be a square that side *s* would like to occupy with his king. Since there are times when both sides can achieve targets, we must quantify the value of obtaining a target. We will assume that if side *s* can occupy the target square, side *s* will queen a pawn in *n* moves, no matter where the defending king is. The value of a target is defined to be the *n* moves before promotion. In general, it is very difficult to locate targets and assign values to them. For a first approximation we will assume that that attacker wins as soon as he captures a pawn. In figure 1, the targets are the black pawns on squares: a5[2], d6, and f5.

There are two possible errors due to this simple-minded scheme for selecting targets which will be considered later. First, targets need not contain a pawn. Consider a passed pawn on the seventh rank. If the attacker can defend the pawn and the queening square without blocking the pawn, then the attacker will queen in one move. Certainly, there are less extreme examples. A second possibility is that the defender may still be able to defend even though the attacker has obtained a target. The most common case is mounting a counter-attack, a more immediate threat than the attacker's. For now, we can ignore these problems since they don't greatly influence our example (figure 1).

Finding Possible Defenses

The first criterion, analogous to a static evaluator, finds all possible defenses, considering only races for targets. That is, for now we will assume that the defender will lose unless his king is as close to every target as the attacking king. We need a few definitions before we can precisely determine the defense sets (dsets.)

Legality

A position is legal if the attacking king and the defending king, on a and d, respectively, are not adjacent and the attacking king is not attacked by a defender's pawn and the defending king is not attacked by an attacker's pawn. (The argument attacker is a binary variable, whose value may be either WHITE or BLACK.)

```
legal (attacker, a, d) ≡
     ~ adjacent (a, d) &
     ~ in-check (attacker, d) &
     ~ in-check (other-side (attacker), a)
```

Distance

We have three kinds of distances, white's distance, black's distance, and the distance on an open board. White's distance between two squares, a and b, is the number of white king moves it would take to move from square a to square b considering pawns as the only obstacles. That is, the white king can move to any square which does not contain a white pawn and is not attacked by a black pawn. Distance on an open board is the number of king moves assuming no obstacles.

Closeness

The attacking king on square a is closer to the target square t than the defending king d iff the attacker's distance from a to t is less than the defender's distance from d to t.

```
closer (attacker, a, d, t) ≡
     distance (attacker, a, t) <
     distance (other-side (attacker), d, t)
```

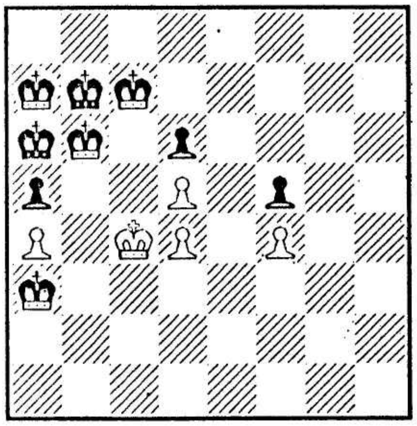
Defense Set

A square d is in the defense set (dset) of an attacking side attacker (either WHITE or BLACK) and a square a iff d is legal for the defender and d is as *close to* every target as a.

```
dset (attacker, a) ≡
     {d | legal (attacker, a, d) &
          $\forall_{t \in targets}$ ~ closer (attacker, a, d, t)}
```

Getting back to figure 1, let us find the defense set of c4 where white is the attacker. c4 is two moves from target a5, three squares from d6 (c5 is illegal for white), and seven from f6. According to this first criterion, black can defend c4 from a7, b7, c7, a6, b6, or a3. (See figure 2 shown below.) In other words, if white's king is on c4, then black's pawns are safe only if the black king is on one of the above defenses a7, b7, c7, a6, b6, or a3. We will see later that most of these defenses are inadequate. Also, we will generalize the meaning of defense to mean that black does not lose, not just that he holds all his pawns.

Figure 2



The black kings mark the elements of the defense set (before triangulation)

The Triangulation Criterion

The second criterion accounts for arbitrarily complex king maneuvers some of which chess players call triangulation, opposition, distant opposition, co-ordination, tempo-ing, or zugzwang[3] by assuring the connectedness condition mentioned above. (We use the subscripts *a* and *d* to denote attacking and defending squares, respectively.) The observation is: if defense sets A and B defend squares a_a and b_a respectively, and if the attacking king can move from a_a to b_a in one move, then the defending king must be able to move from any square a_d in A to some square b_d in B in exactly one move. Squares not meeting this constraint, hereafter called the triangulation criterion, are removed from the defense sets.

> A side can move from square *a* to square *b* if the two squares are adjacent and the destination square *b* is legal for the side *attacker* assuming that the defending king is on square *dking*.

```
can-move? (attacker, a, b, dking) ≡
     adjacent (a, b) & legal (attacker, b, dking)
```

Given two attacking squares, a_a and b_a, and their respective defense sets before triangulation, A and B, we want to find the new defense sets after triangulation. We need two functions to manipulate defense sets, FETCH-DSET and REMOVE. The function FETCH-DSET retrieves the defense set of a given attacking square from the map of co-ordinate squares. The REMOVE function removes a given defense from a given dset with side effects.

```
triangulate:proc (attacker, $a_a$, $b_a$)
     A := fetch-dset (attacker, $a_a$)
     B := fetch-dset (attacker, $b_a$)
     defender := other-side (attacker)
     for all $a_d$ in A do
          if $\sim\exists_{b_d \in B}$ [can-move? (attacker, $a_a$, $b_a$, $a_d$) →
                    can-move? (defender, $a_d$, $b_d$, $b_a$)]
               then remove ($a_d$, A)
```

Returning to our example (figure 1), let a_a be c4 and b_a be b5. Then A = {a7, b7, c7, a6, b6, a3} (as previously calculated) and B = {} according to the first criterion. Does a7 still defend c4 after triangulation? Since white is threatening to move from c4 to b5, all defenses to c4 must either prevent white from moving to b5 or must be adjacent to some defense of b5. Since there are no defenses to b5, all defenses of c4 must prevent white from moving to b5. In particular, a7, failing to keep white out of b5, is an inadequate defense of c4. Similarly, b7, c7, and a3 are removed from A. At this point A = {a6, b6}.

It should be apparent that defenses depend upon neighboring defenses. After having heavily reduced the defenses to c4, we should suspect that the reduction might propagate. For example, a8 defends against b3 before triangulation. Since white can move from b3 to c4, black must be able to move from a8 to some defense of c4. Before we eliminated a7 from the defense set of c4, black could move from a8 to a7. We can see that reducing the defense set of c4 causes the defense set of b3 to shrink.

This propagation effect explains why a6 is eventually removed from the defense set of c4. Notice that there is no defense to h5. Consequently, the only defense to h4 is g6. Defenses to g3 must be on the f file. Continuing this reasoning, the algorithm discovers the constraint given in Fine, the black king cannot allow the white king to be two files closer to the king side. If black attempted to defend c4 from a6, he would lose his pawn on f5.

Selected Results

white attacking square	black defense set
b5	{}
c4	{b6}
d3	{c7}
c3	{b7}
b3	{a7, c7}
d2	{c8}
c2	{b8}
b2	{a8, c8}
d1	{c7}
c1	{b7}
b1	{a7, c7}
a3	{b8, b7, d7, d8}
a2	{b8, b7, d7, d8}
a1	{b8, b7, d7, d8}
...	

The extremely small size of the defense sets illustrates just how constrained black really is. A similar calculation with white as the defender yields much larger sets, as would be expected.

With white to move, the above map tells us that black cannot co-ordinate with white because it is after black's move and the black king on a7 does not occupy a square in the defense set {b8, b7, d7, d8} of the white king on a1. If black were on b8, b7, d7, or d8 the game would be drawn. However, since this is not the case, black is lost. White can demonstrate the win only with Kb1, anything else would be a grave mistake, allowing black to co-ordinate. Let us consider the other possibilities, Ka2 and Kb2. Ka2 could be answered with either Kb7 or Kb8. After the other possibility, Kb1, black must play somewhat sharper, and only Ka7 will save the game.

On the other hand, with black to move, black can co-ordinate, drawing the game, with either Kb7 or Kb8. Neither Fine nor Averbakh suggest Kb8 because that defense is somewhat "anti-theoretical" meaning that it gives away the twisted distant opposition[4].

The Triangulation Criterion Viewed as a Waltz Filter

Waltz [Winston, chapter 1] developed a filtering process to search a line drawing scene. The problem is to find all possible ways to label the lines (- for concave, + for convex, and either ← or → for a boundary) in a scene so that lines meet in physically realizable vertices. Waltz's algorithm first assigns each vertex a set of possible labels, i.e.

physically realizable. The next and final step is the filtering process which assures that connected vertices are labeled consistently, that two vertices connected by a line are labeled the same way at both ends. The procedure is to iterate through the vertices checking each possible label for consistency with each connected neighbor. If any possible label is found to be inconsistent with a neighbor, that label is removed from the possible labels. It is possible that the effects of removing a label might propagate to other vertices. After every removal, the possible propagations are immediately considered.

The triangulation procedure is remarkably analogous to Waltz's filter. The vertices correspond to the possible locations of the attacking king. A defense set can be thought of as a set of physically realizable ways to label a vertex. Two attacking squares are connected iff the attacker can move from one attacking square to the other. Just as two vertices connected by a line must be labeled consistently, the defense sets of two attacking squares must be consistent. That is, the defender must be able to move from any square in one defense set to a square in the other defense set.

This analogy shows us that we can apply the triangulation constraint in time proportional to the number of vertices (~64) times the number of possible labels (~64), which is pretty good. Actually we can probably do even better.

How many comparison operations, in this case applications of the triangulation criterion, must we perform? They must be performed until the graph, i.e. map of co-ordinate squares, has "settled", in that future comparison operations will not shrink a defense set. At this point we say that the graph is stable. It is easy to show that the procedure will terminate. A comparison operation can either leave the defense sets alone or shrink them. Since the sets are of finite size, only finitely many removal operations can be performed. If all sets are empied, no futher removals will shrink them, and the graph is therefore stable, so the algorithm halts. Hopefully, though, the map will settle before every element is removed.

The time to triangulate is related to the number of comparison operations required. Waltz has given us the upper bound discussed above. However, in general, it is possible to improve upon this upper bound by comparing the sets in the optimal order. The most constrained sets should be compared with their neighbors first because they are most likely to shrink their neighbors. We can assure that the most constrained sets will be considered first if we keep a sorted queue of sets to be considered.

How do we decide if one set is more constrained than another? The best answer is to employ some domain restricted knowledge. For example, we might use a heuristic which suggests that attacking squares closer to targets tend to be more constrained. However, in this work, we have chosen a much simpler and domain independent ordering heuristic. Given two vertices, *a* and *b*, associated with possible labels sets *A* and *B*, respectively, we define *a* to be more constrained than *b* iff set *A* is smaller than set *B*.

The queue improvement leads to some other optimizations. In Waltz's filter, the propagation possibility is checked immediately after a set is reduced. An improvement is to re-queue a vertex after the associated possible labels set has been reduced. This assures that the vertices will be considered in the proper order. By checking for propagation immediately, this order cannot be guaranteed.

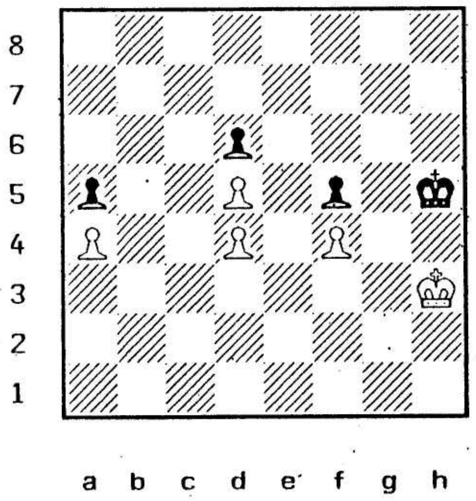
Another advantage of the queue is that it can avoid considering some vertices multiple times. Imagine that a comparison operation shrinks a defense set which is already on the queue. Instead of queuing the vertex twice, the vertex is simply re-ordered on the queue. If we didn't have the queue to remember which vertices must be looked at, then we would have to compare the vertex against its neighbors twice. It is very difficult, though, to see exactly how good the queue is.

Bottleneck Defenses

An important observation is that the triangulation heuristic correctly weeds out most mistaken defenses generated by the first criterion as long as they are sufficiently far from targets. Erroneous defenses close to targets might survive triangulation since there is a greater chance that the erroneous defense will be next to a correct defense. At any rate, it is easy to find examples where triangulation corrected an error in the first criterion. One such example is that b7 doesn't even defend c4 in a simple race and yet, b7 is included in the defense set of c4 until the triangulation heuristic removes it. This insight leads us to the conclusion that only squares near targets must be calculated carefully. Including too many squares sufficiently far from targets in defense sets will be corrected by the triangulation.

In particular, the reason that b7 is included in the defense set of c4 is that the route from b7 to a5 passes through a bottleneck (b6, a6) which white can seal off with Kb5. Fortunately, in this case, the bottleneck caused too many squares to be included in a defense set. However, this need not always be the case. For example, what is white's defense set when black's king is on h5? (When we are looking for white's defense set, black is considered the attacker.)

Figure 3



Why isn't black's king closer to d5?

For the white king to get to d5, white has to move all the way around his d4 pawn (h3-g3-f3-e3-d3-c4-d5), a total of six moves. Since there are no pawn obstacles hindering the black king, black can reach d5 in only four moves (h5-g4-f4-e4-d5), long before white can defend it. The algorithm deduces that black is closer to d5 than white.

The reason black isn't closer to d5 is that black cannot break through the bottleneck on g4 and h4, black's only entrance to white's territory and white's pawns. How do human chess players almost immediately "see" that the bottleneck holds? There are two components to a bottleneck defense. First, we must suspect that a bottleneck defense might work and secondly, we must check that it is connected to other defenses. The later connectedness condition is our familiar triangulation criterion. Suspecting a bottleneck defense is somewhat more difficult. After considerable experimentation, we have arrived at the following quite satisfactory heuristic.

> A possible bottleneck defense (before triangulation) exists when the defending king d is in a position to block a bottleneck square on the path from the attacking king a to a target t. To do this, the defending king must occupy a square which is, from the attacker's point of view, no farther from the target than the attacker's king. Secondly, the defending king must be between the attacking king and the target. In other words, using what mathematicians call the triangle inequality, the distance between the kings must be strictly less than the distance from the attacking king to the target.

```
bottleneck (attacker, a, d) ≡
     distance (attacker, d, t) ≤ distance (attacker, a, t) &
     distance-on-open-board (d, a) < distance (attacker, a, t)
```

> A bottleneck defense is an alternative defense to the "closer" defense previously discussed.

```
dset (attacker, a) ≡
     {d | legal (attacker, a, d) &
          ∀_{t ∈ targets} [~closer (attacker, a, d, t) ∨
                           bottleneck (attacker, a, d, t)]}
```

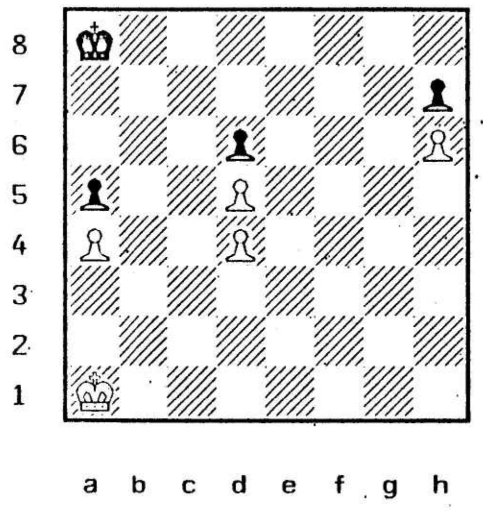
In figure 3, h3 is a bottleneck defense because the white king on h3 is holding a square which is just as many black moves from d5 as the black king on h5. Also, white's king is between the black king and the target on d5 since the distance between the two kings is only two whereas the black king is four black moves from d5.

This definition of bottleneck defenses is somewhat more heuristic than many others in this paper. What sorts of errors might result and how might they be corrected? When the attacking king is very close to the target it is impossible for the defender to be in between the attacking king and the target. At large distances, though, our bottleneck definition can suspect some inadequate defenses. For example, let white be the attacker. Even though c8 does not defend a8, the heuristic given above labels c8 as a possible bottleneck defense to a8. The triangulation criterion will weed c8 out because black has no defense to white's straight forward threat a8–a7–a6–a5.

Counter-Attack Defenses

A much more serious restriction with the above algorithm is that it does not yet consider counter attack defenses.

Figure 4



drawn (either side to move) [Averbakh, problem #673]

Consider the above problem with the white king on f5 and the black one on f7. If white races from f5 after the black pawn on a5 he will win black's pawn. Consequently, the f7 is not in the defense set of f5. The problem is that if white should attempt to capture the a5 pawn, black could launch a counter attack on white's h6 pawn, winning the game.

These counter attacks are yet another alternative to the passive defenses already considered when creating defense sets. A square is a counter attack defense if capturing one of the attacker's targets allows the defender to queen a pawn in at most one move after the attacker will queen one of his. In other words, this covers the case where the kings race off to different sides of the board. (It does, however, avoid complications of evaluating the queen endgames that might result.)

```
dset (attacker, a) ≡
     {d | legal (attacker, a, d) &
          $\forall_{t \in \text{targets}}$ [~closer (attacker, a, d, t) ∨
                    bottleneck (attacker, a, d, t) ∨
                    counter-attack (attacker, a, d, t)]}
```

It is rather difficult to give a precise definition of a counter attack defense because it is implemented with a rather large number of heuristics. Basically, there is a counting procedure which computes the number of moves to capture a pawn target which is blocking a friendly pawn, move out of the way of the friendly pawn, and queen that pawn.

In the figure 3 with the white king on f5 and black's on f7, white can capture black's a5 pawn in 5 moves and queen 5 moves later. Black's defense is that he can capture white's h6 pawn in 2 and queen in 6 more. With the counter attack defenses, not only does f7 defend f5 but so do a4 and a host of other squares well inside white territory. Although the defense sets are no longer so attractively small, the core of usually obtainable squares is only slightly larger than before. The fact is that many squares well inside of white's territory are perfectly good defenses. We will discuss methods to eliminate unreachable defenses in the section called Forward Pruning.

An additional problem with counter attack defenses is that triangulation, as previously defined, tends to remove the counter attack defenses. Consider the above problem with the white king on c4 and black king on h6. Having captured the pawn on h6, black has obtained his target position. To win the game, though, black will have to play some pawn moves to queen his h7 pawn. The problem is that triangulation assumes that all good positions are connected with only king moves. We can see that although h6 defends c4, there is a square adjacent to c4, namely b5, which cannot be defended by any square adjacent to h6. This shows that counter attack defenses, unlike passive defenses, are discontinuous. Once h6 has been captured, the black king cannot make any more progress toward a white target. This problem has been solved in the current implementation by preventing triangulation from removing a counter attack defense from a defense set if the defending square is closer to the counter attack target than the attacking king. Not all counter attack defenses are protected from triangulation, only those where the defending king is guaranteed to reach the counter attack target no matter what the attacker does. In our example (figure 4), f7 could be removed from the defense set of f5 (both f7 and f5 are 2 from h6) but it isn't because g6 cannot be removed from the defense set of e4 (g6 to h6 is 1 but e4 to h6 is 3.) This is the first case where king moves and pawn moves are coupled.

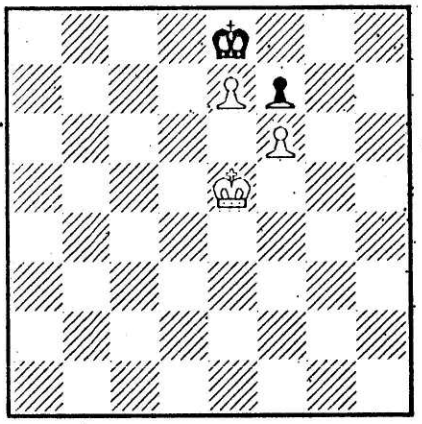
b5	{h6}
f5	{e7, f7, h6, a4, b4, a3, b3}
g5	{e7, f7, h6, a4, b4, b3, a3}
c4	{b6, g6, h6, a4}
e4	{d8, e8, f8, g8, d7, e7, f7, f6, g6, h6, g5, h5, a5, b4, c4, g4, a3, b3, a2, b2, c2}
b3	{a7, c7, f7, f6, h6, d5, f5, h5, g5, h5, d4, e4, f4, g4, h4, d3, e3}
c3	{b7, f7, f6, g6, h6, d5, f5, g5, h5, a4, e4, f4, g4, h4, a3, e3}
d3	{c7, f7, f6, g6, h6, d5, f5, g5, h5, a4, b4, f4, g4, h4, a3, b3, a2, b2}
b2	{a8, b8, c8, e8, f8, g8, e7, f7, f6, g6, h6, d5, f5, g5, h5, a4, b4, c4, d4, e4, f5, g4, h4, d3, e3, f3, g3 h3, d2, e2, f2, d1}
c2	{b8, c8, e8, f8, g8, e7, f7, f6, g6, h6, f5, g5, h5, a4, b4, c4, d4, e4, f4, g4, h4, a3, e3, f3, g3, h3, a2}
...	

Apparently the defense sets are not nearly as constrained as in problem Fine #70 because black can defend the king side entrance from e7 instead of g6.

Consequently there isn't the long propagating constraint from the king side. Notice that the defense set of b2 and that of c2 overlap on two adjacent squares, b8 and c8, a strong indication of a drawn position. To draw, all black has to do is get to either b8 or c8 when white is on either b2 or c2. Once he has reached one of these squares, he can move between them forever. It is the attacker's responsibility to break an infinite repetition. It is easy to verify that the position is indeed drawn by completing the rest of this map.

Stalemate Defenses

Figure 5



White cannot make progress by moving toward the queenside. [Fine, problem 29]

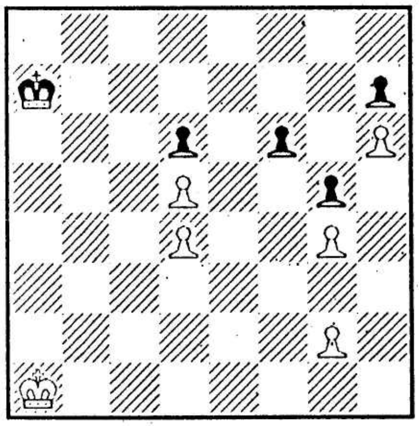
If white moves to d6, c7, or c8 black will be stalemated, a rather sufficient defense. Obviously, the first criterion must know what stalemate is and consider stalemate as yet another alternative to the closeness, bottleneck, and counter attacker defenses. It is worth noting that these stalemate defenses can have more global effects because of the triangulation criterion.

Pawn Moves

Now that we have discussed king co-ordination, it is time to consider how pawn moves might affect the kings. We have actually already considered some coupling between pawns and kings. Stalemate depends upon a complete lack of pawn moves. Also, the reason king moves are discontinuous in counter attacks is that after the defending king has captured the target, the defender must finish the counter attack with pawn moves. So, in fact, we have discussed some pawn moves. In this section, though, we would like to explictly mention some other important themes centered about pawns.

Pawn Moves to Gain Tempo[5]

Figure 6



white to move wins; black to move draws [Averbakh, problem #676]

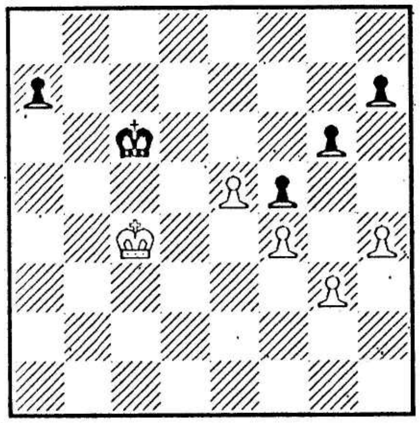
When the attacker has an extra tempo, he may consume the tempo at an optimal time, forcing the defender to play. Intuitively, the tempo is an option to pass. The solution is to build one map of co-ordinate squares assuming there is no pawn tempo. Construct another map using the constraint that for any square a_i in a defense set A, there must be an a_2 in A such that a_1 is adjacent to a_2. After applying triangulation to the new map of co-ordinate squares, we have the map which accounts for the tempo. To illustrate that this approach does in fact work, some selected results are shown below.

f5	{e7}
a4	{b6, d5, c4, d4, g4, g2}
b4	{a6, d5, d4, g4, g2}
c4	{b6, g4, g2}
e4	{d8, d7, g4, g2}
a3	{b7, d5, c4, d4, e4, f4, g4, h4, c3, d3, e3, g3, f2, g2, f1, h1}
b3	{a7, d4, f4, g4, h4, d3, f2, h2, f1, g1, h1}
c3	{b7, f4, g4, h4, c3, g3, f2, g2, h2, f1, g1, h1}
d3	{c7, f4, h4, g3, f2, g2, h2, f1, g1, h1}
a2	{b8, d5, a4, b4, c4, d4, e4, f4, g4, h4, c3, d3, e3, g3, c2, d2, e2, f2, g2, h2, e1, f1, g1, h1}
b2	{a8, b4, c4, d4, e4, f4, g4, h4, d3, ...}
c2	{b8, b4, ...}
d2	{c8, d4, e4, ...}
a1	{b7, a5, b5, d5, b4, c4, d4, ...}
b1	{a7, b5, d5, b4, c4, d4, ...}
c1	{b7, d5, c4, ...}
d1	{c7, d5, ...}
...	

We can see that white can win by Kb1 and black can draw with Kb7.

Passed Pawns

Figure 7



White to move and win [Fine, problem #88]

A pawn p is passed if the opponent does not have a pawn on a rank between p and the queening square on the same file as p or on an adjacent file. Both a7 and e5 are passed in the above problem. Clearly, these pawns must affect the co-ordinate squares since, if one king should leave the *square of a passed pawn*[6], the region where the king can catch the pawn running toward the last rank, then the pawn will promote into a queen. Also, these pawns affect the targets. Suppose the white king is on f7. There is no place the defending king can be and prevent white from queening in only three pawn moves.

First, let's consider the targets. If a pawn is passed, then the squares on adjacent files and two ranks ahead of the passed pawn are targets. If the passed pawn is on the seventh, then the targets are only one rank ahead of the pawn. In our example (figure 7), we add f7 and d7 to the list of white's targets and b5 to black's targets.

The decision to make b5 a target has some problems. If black can occupy b5, then he is guaranteed of making progress with his a7 pawn. However, he is not guaranteed of queening the a7 pawn in a fixed time period. Similarly, a7 is not as valuable a white target as the others since reaching it does not obviously guarantee a pawn promotion. The notion of target needs some work.

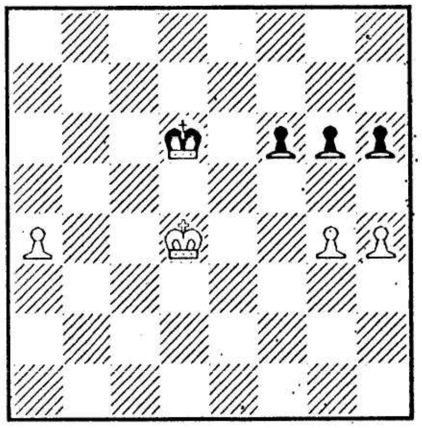
Nevertheless, while we are on the subject of targets, it is clear that some squares occupied by a pawn are not targets. In particular, if the pawn is protected by another pawn, then there is no way to win the target. It is a waste of effort to include such protected pawns in the target list. Similarly, if a pawn is protected by a sufficient threat, such as a passed pawn threatening to queen extremely quickly, then the pawn is also not a target. For example, g3 is not a black target since white would queen his e5 pawn long before black could do anything from g3. In this problem, black has no targets and the white targets are a7, d7, and f7.

To consider the effect of the square of passed pawns we have to be more precise about what a target means. Once a target has been captured, the attacker will queen a pawn in certain number of additional moves. For example, white will queen three moves after he reaches the target on f7. Sometimes capturing a target does not clearly lead to the queening of a pawn. In this case, the target is associated with a large time. The target a7 is a good example of a target which does not explicitly block a friendly pawn. However, since capturing it does free something and does eventually lead to a pawn promotion, we consider the time before such a promotion to be large but not infinite.

Computing the number of moves before queening is also part of the counter attack calculation. As in the counter attack calculation, we have to compute the time for the attacker to queen a pawn by force and the time for the defender's threat. If a king is outside the square of a passed pawn, then the time to queen an attacking pawn has an upper bound of the time to queen the passed pawn. It turns out that this is sufficient knowledge to understand problem 88.

Deflection Sacrifices

Figure 8



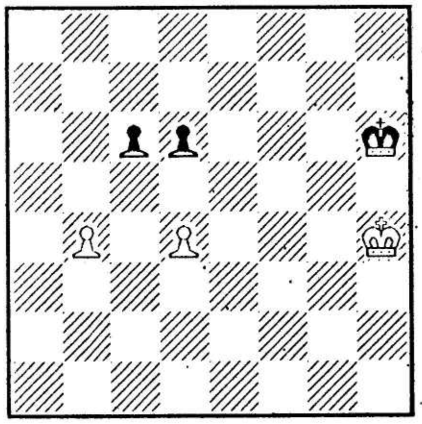
White to move and win [Fine, problem #62]

White's winning idea is to push his outside passed a4 pawn forcing black's king over to the queen side. With black's king out of play, white can easily pick up black's king side pawns.

How might this idea fit into our way of thinking? Whenever the attacker has a passed pawn, the defending king can be forced over to catch it. We can calculate exactly which square the defending king will be forced to. The pawn, before being captured, will advance as many moves as there are files between the pawn and the defending king. If the defending king cannot defend the position from where he captures the passed pawn, then the pawn sacrifice works. In other words, the defending king, after capturing the sacrificed pawn, must have at least a plausible defense (first criterion) in order to defend against the deflection sacrifice.

Potentially Passed Pawns

Figure 9



White to move and win [Fine, problem #51]

White's winning idea is that his b4 pawn is potentially passed and so far outside that black's king cannot stop it. The difficulty of this problem is realizing that the pawn on b4 is potentially passed, that there is a series of moves that makes b4 passed by force.

One solution is a branching tree of pawn moves. Since there are rarely many legal pawn moves, the tree is sufficiently bounded. Unfortunately, although the tree correctly finds the potentially passed pawns, it does not tell us anything about possible king intervention. For example, if the white king were on h8, then the pawn sacrifice leading to the potential passed b4 pawn would not work. The white king must be in the square of d5 to win this problem.

An important observation is that there are not very many different ways to create a passed pawn. The solution taken here is to model the different situations with a set of patterns. For each pattern, there are associated experts which can compute the effects of different king positions. Should a pattern lead to an unstoppable pawn, the expert will return the number of pawn moves before the pawn queens. Unless the defender can generate a counter attack before the pawn queens, he will lose.

What should these patterns look like? We would like to exploit as many symmetries as possible in order to limit the number of patterns. The left-right symmetry can be easily exploited. Much more importantly, it is possible to split the pawns into *pawn clusters*, independent groups of pawns. The reasoning behind pawn clusters is that pawn moves are extremely local; pawn moves in one cluster do not affect other clusters.

Pawn Cluster

> A pawn cluster consists of two sets, one of attacking pawns and one of defending pawns. The attacking pawns are separated by no more than one file. If there is a file without an attacking pawn, then that file must contain a defending pawn. Otherwise there would be two pawn clusters. The defending pawns are all pawns on the same file or a file adjacent to an attacking pawn.

The most important key on a pawn pattern is the number of attacking pawns and the number of defending pawns in a cluster. Other features which key certain experts are the ranks of the pawns, backward pawns, doubled pawns, pawns split by a single file, locked pawns, protected pawns, and half passed.

The above problem, Fine #51, contains a single pawn cluster. There is an expert which classifies the cluster into the two on two split attacking pawn category. An expert in this category knows about the "split pawn sac" which forces a pawn promotion in four moves. Two more difficult examples, Fine #61 and #82, are shown below in the evaluation section.

Most clusters of three or fewer attacking pawns have been worked out. More work is necessary to consider the larger clusters.

Co-ordination Compared with a Branching Tree

In this section we wish to compare the co-ordinate squares procedure as discussed above with the more standard depth first branching tree procedure exemplified by such machines as Peasant. This section is not intended to determine which approach is the best. We merely plan to explore their relative merits and trade-offs. It is our belief that co-ordinate squares, as a generalization of Waltz's filter, is important to Artificial Intelligence, even if there is a better approach to solving the same problems.

It will be assumed that the branching tree strategy employs a hash table to remember all the nodes previously visited. (In fact, part of the problem with Peasant is that it, for some inexplainable reason, removes many entries from the hash table.) In many ways, the hash table is analogous to the co-ordinate squares map. Both data structures list every position considered associated with a value for the position. In a hash table representation the value is whatever was passed back up the tree whereas the value in the co-ordinate squares representation is whether the position is an adequate defense. Just as we have considered the co-ordinate squares map to be the understanding of a chess position, the hash table can be considered the results of a branching tree search. It is possible to determine easily the best move using either data structure. Furthermore, there is an argument that both searches require roughly equal time because the limiting factor in all of these endgames is the small number of distinct positions resulting after an arbitrary series of moves. If both searches will eventually find all of these distinct positions then it really doesn't matter in which order they are searched.

Although it is very difficult to refute the above argument, the work here is based upon the intuitive belief that in a practical situation a branching tree will explode given half a chance. The co-ordinate squares approach restricts the number of positions to be considered before starting whereas a branching tree discovers that the number of positions is limited only after the hash table contains all of these positions. Restricting the number before the search has the advantage of preventing any chance of an explosive search. However, it may suffer by failing to consider something important.

Perhaps the greatest advantage with the co-ordinate squares approach is that the order of evaluating each node has not yet been specified. In a branching tree algorithm, the order must be depth first in order to employ alpha-beta pruning. In the co-ordinate squares implementation, triangulation considers the smallest defense sets first. In the first example, Fine #70, the defense sets of b5, g5, d3 and c4 would be considered long before the defense sets of a2, b1, and b2.

This, though, leads to a serious drawback, co-ordinate squares cannot use a number of pruning techiques used in branching trees such as alpha-beta. Since the number of positions is sufficiently limited, failing to prune a number of these positions means that the process will take longer, not that it will explode.

A much more minor difference is that the map of co-ordinate squares needs less space than a hash table used in a branching tree strategy because the hash table must contain, in addition to the information in the co-ordinate squares map, a token indicating whether or not a particular node is in the hash table. This token is implicitly contained within the co-ordinate squares map since the map is complete. In addition, not having to check the token represents quite a savings if one considers the number of hash matches, different series of moves resulting in the same position, there are in these problems.

Since the co-ordinate squares map is complete, we don't have to recalculate the entire map after each move is played. No matter what move is played in the game, the new position must have been considered in the co-ordinate squares map. The machine can find the co-ordinating moves almost immediately from the old co-ordinate squares map. In a tournament situation, this machine would take a long time as soon as the king and pawn endgame was entered and every move afterward would be played extremely quickly. Most good human chess players spend a relatively long time planning and then play a series of moves very quickly.

Saving a co-ordinate squares map is analogous to saving a hash table, which is rarely done in branching tree strategies because the number of nodes, in general, will be too large. The number of nodes is sufficiently small in this work partly because of the nature of king and pawn endgames and partly because the algorithm described here went to great lengths to limit the number of nodes to the different ways to place the kings.

Another advantage of co-ordinate squares is that the triangulation calculation can be optimized to run much faster than the analogous branching operation. For example, the defense sets can be represented as a 64 long bit string. The inner loop of the triangulation calculation on two sets A and B could be implemented by generating the set of squares adjacent to the elements of set A and intersecting the result with set B. Using this representation, intersection requires one AND operation. Similarly, the generation of the adjacent squares can be implemented with a fair amount of parallelism[7]. On the other hand, it is considerably more difficult to implement parallelism in a forward reasoning strategy, using traditional computer hardware.

The most serious disadvantage with co-ordinate squares is that the number of terminal nodes must be extremely limited and that each one of these must be superficially considered by the first criterion. A branching tree can avoid considering unobtainable defenses with pruning techniques such as alpha-beta. To restrict the number of nodes to the different ways of placing the kings (~ 4000) so that co-ordinate squares would be practical, the solutions to the above problems employed a number of domain specific tricks, such as tempo and counter attacks. A branching tree is more general and may even be faster if most nodes are unobtainable. Clearly there are times when one of the approaches, co-ordinate squares or branching tree, will be more appropriate. The perfect chess machine may have to understand both and understand the trade-offs.

Forward Pruning

One complaint with co-ordinate squares is that quite a number of unreachable positions are considered. In this section, two heuristics will be discussed to reduce the number of positions that we must consider. Experimentally, these heuristics represent a fifty percent time savings.

The first heuristic is based upon the observation that the defending king must be able to reach a defense before the attacking king can reach the attacking square. Although this idea seams perfectly obvious, some care must be taken. First of all, this heuristic changes the meaning of a defense. Before, a defense set contained all defenses. Now the defense sets contain only those defenses which the defender can reach. It is important to consider this subtle point when examining these new defense sets.

A possible source of error may arise if this heuristic removes too many defenses. Even though the defender may not be able to reach some defenses if the attacker moves directly toward an attacking square, the defender may need one of these defenses if the attacker should triangulate, take a longer path so the defender will be on the move. It is very difficult to generate an example because cases are so rare and are usually quite complicated. However, it is necessary to compute defenses just outside the area the defender can actually reach.

A second observation is that the attacker should not consider moving to every attacking square. It is a waste of time for the attacking king to run away from all the targets. We will only consider attacking squares which make progress. That is, if square a is farther from the initial attacker's king position than square b, then a must be as close to some target as b in order to make progress. Again, experimentally it was discovered that this should be relaxed for squares adjacent to the initial placement of the attacking king since the initial position may already be so close to some target that the only way to make progress is to back up temporarily.

Certainly both of these ideas are highly heuristic; they neither catch all unreachable defenses nor are they completely correct. However, for our purposes, the time savings outweighs the possible errors that may result.

Cost Analysis

The space requirements of the co-ordinate squares map is rather minimal. We can represent each defense set as a 64 bit string, one bit for each square. If the i^{th} bit is set then square i is in the defense set. With this notation, the entire co-ordinate squares map requires only 64^2 bits.

The time costs are somewhat more difficult to analyze because the author does not fully understand the effects of several time saving heuristics such as the queue modification to Waltz's algorithm or the forward pruning conditions. At any rate, the time cost is due mainly to two factors, the time to apply the first criterion to each position and the time to apply the triangulation criterion.

The first has an upper limit which is linear with the number of positions being considered using the straight forward approach, literally applying a static evaluation to each position. It is possible to do even better if the static evaluator is organized in a more sensible way, keyed on themes instead of positions. For example, we know that that defending king must be in the square of an attacker's passed pawn. (We won't consider counter attacks at this point.) One way to implement this constraint is for the static evaluator to try all placements of both kings, testing to see that the defending king is in the square of the attacker's pawn. A more intelligent approach is to design a procedure which generates those squares within the square of the passed pawn. Only these squares would be considered as possible defenses. This second approach is probably faster than linear with the number of positions being considered.

The time to triangulate also has an upper limit which is constant with the number of positions being considered. The argument, as outlined in the section relating the triangulation criterion to Waltz's algorithm, is that the time is linear with the number of vertices (attacking positions) times the number of labels (defending positions.)

In practice, the co-ordinate squares program consumes about 110K words on a PDP-10. Problems require one to ten minutes of cpu time, most typically about five. When the defending king is lost in the initial position according to the first criterion, the program halts without considering triangulation. These problems take only one minute, the time to initialize the data structures, find targets, and measure distances from the targets to every square. The most expensive problems have a number of potentially passed pawns. The current implementation checks all the possible pawn sacrifices to create a passed pawn for each position. A major improvement would be for the experts which know about pawn clusters to return the constraints on the two kings. Currently, they have to be called for each different placement of the kings.

At any rate, this performance demonstrates that co-ordinate squares is sufficiently efficient for tournament play. Once the co-ordinate squares map has been found, every move for the rest of the game can be played almost instantly. A ten minute time investment is quite acceptable. Also, the co-ordinate squares map can be drawn during the opponent's move. Of course, since order of magnitude improvements in computer performance can be expected, this ten minute cost could easily be reduced to one in the not too distant future. There are existing machines which are almost an order of magnitude faster than the KA PDP-10 currently being used. (After this document had been completed, we converted the code to run on the new LISP MACHINE currently being developed in the AI lab at MIT. This version solves most problems in less than a half minute of real time.)

Future Work

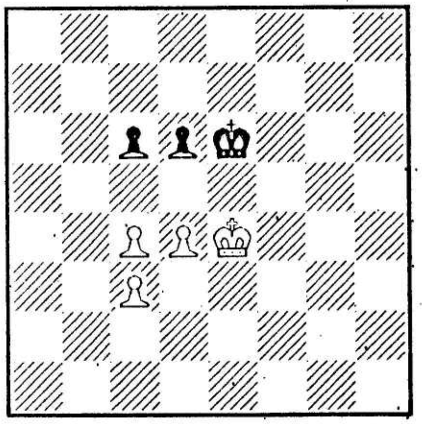
There is still a considerable amount of work to be done. In this section, we will list a few common themes either completely missing from the existing co-ordinate squares program or desperately in need of refinement. Most of these involve pawn moves since the coupling between king moves and pawn moves is not well understood.

Shifting Square

Every passed pawn has a square, a region where the defending king could still catch the pawn before it queens. If the attacker has two passed pawns, though, the two squares are coupled in a very strange way. As the defending king attacks the closer passed pawn, the other one advances reducing its own square. If the defending king should continue after the first passed pawn, he may end up outside the other one's square. If the defending king should attack the second passed pawn, then the first can advance thus protecting the first. Some simple cases of this have been implemented.

Pawn Moves which Change the Critical Squares

Figure 10



White to move and win [Fine, problem #58]

White can win by pushing his d4 pawn. If black trades pawns, it is easy to show (with techniques developed in this paper) that black is lost. Should black retreat his king to d7, white can trade pawns himself. Using zugzwang, white can occupy d5. Once on d5, white can advance the c4 pawn to c5. After black trades d6 for c5, white has a won king and single pawn ending.

The techniques developed in this paper fail to understand a number of the ideas involved in the above solution. First of all, when a pawn moves it attacks different squares. Consequently, some squares the kings used to be allowed on are now off limits and some squares that had been off limits are now legal. Also, moving the pawns can move some targets. For example, after 1. d5, c6xd5, the target on c6 is gone.

What is missing is an understanding of how pawn moves can change targets and co-ordinate squares. Since there are rarely more than a few pawn moves and even fewer are sensible and fewer yet can change critical squares, it should be possible to identify these rare but important pawn moves. However, these pawn moves are the most difficult area for future research.

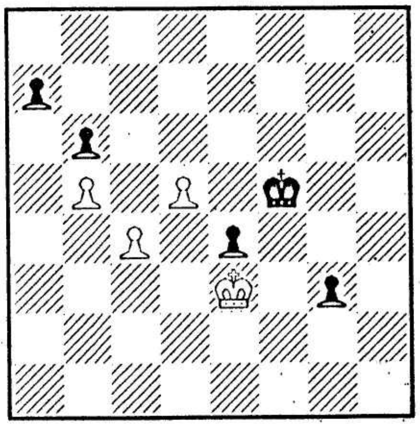
Single Pawn Endings

Another reason that this work does not solve the above problem is that the algorithm does not know which trivial single pawn endings are won and which are drawn. There is no machinery to make the final deduction, that after black trades d6 for c5, the single pawn ending is won for white. Since these endings are not difficult and have been completely understood by Peasant [Frey, chapter 5] and others [Tan] [Perdue], it should not be too difficult to add the lacking knowledge.

Undermining Pawn Chains

One solution to figure 11 (below), credited to Grigoriev, is to bring the black king back to b7 and advance the a7 pawn, forcing a pawn trade. After the pawn trade, white's d5 pawn, no longer protected by a pawn, can be captured. Again, this there is nothing particularly difficult about this theme. Nevertheless, it has not been implemented.

Figure 11



Black to move and win [Averbakh, problem #710]

To conclude this section, we must feel that there is a considerable amount of work to be done before a machine will play endgames perfectly or even consistently better than good humans. However, the fact that we can even list the most important work to be accomplished is very encouraging. Winston often talks about the "small size of infinity," observing that relatively few themes can accomplish a complicated task which, before serious study, appeared to involve almost an infinite number of themes.

Evaluation

The author of *Peasant*, Monroe Newborn, chose to evaluate his program with sixteen problems selected from Fine. Although *Peasant* finds the correct move in eleven of them, it appears to see the win in only six.. A number of the correct moves are forced since all other moves lead to immediate disasters. There is a significant difference, though, between avoiding short range disasters and finding a long range winning theme.

Four of these problems have already been discussed, problems 29, 51, 58, and 70. The first two are well understood by both programs, neither understands problem 58, and problem 70 requires co-ordinate squares. Peasant is much faster on problem 29 because it is relatively shallow, 11 plies. Even more importantly, since black can occupy only two squares, d7 and e8, alpha-beta cutoffs are much more useful than usual. Co-ordinate squares, as currently implemented, considers quite a number of ridiculous black defenses. On the other hand, Peasant couldn't solve problem 70 because it requires about thirty plies.

Figure 12

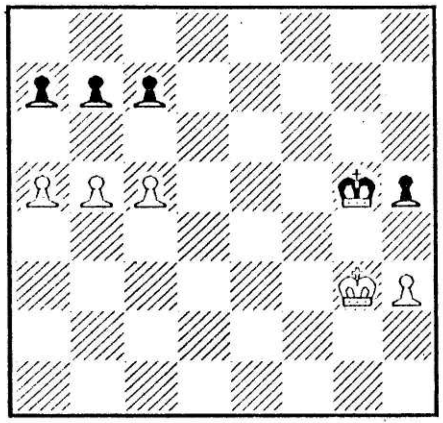
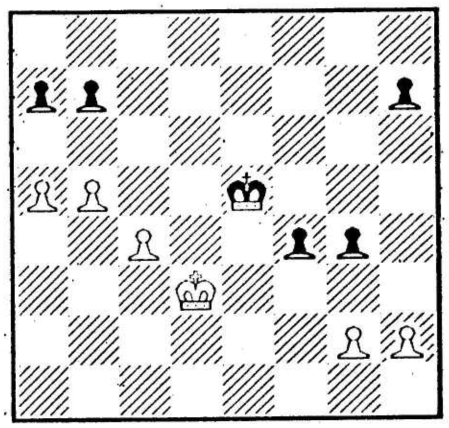

problem 61
white to move and win

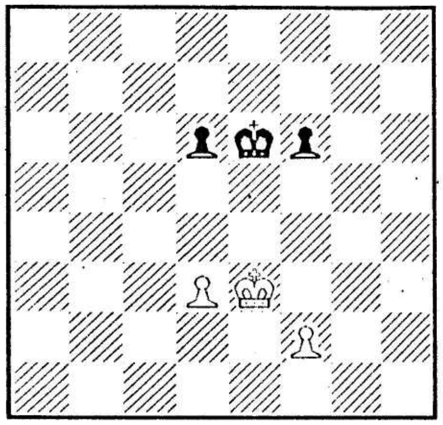
Figure 13


problem 82
white to move and win

Both of these problems involve a potential passed pawn queening before the defending black king can catch it. Peasant, lacking the useful knowledge regarding potential passed pawns, has to carry out the general tree search. It can't find the solution to problem 82 in eight plies since black has some delaying moves on the king side. Our program solves both of these in just about one minute of cpu time.

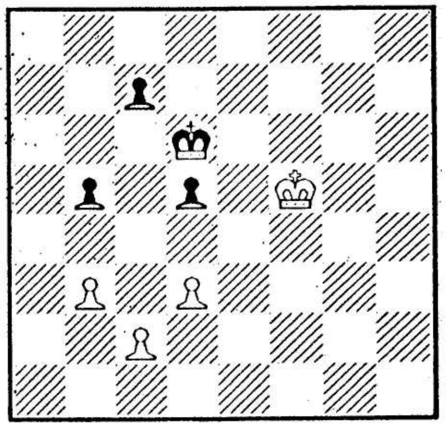
The solution is found by a pattern match on the queenside pawn cluster in both figure 12 and figure 13. The expert associated with each of these two patterns knows a number of stero-typical sacrifices to consider. The winning sacrifice in figure 12 is: 1. b6, c7xb6; 2. a6 and white has a winning passed pawn. Similarly, the solution to figure 13 is: 1. c5, Kd5; 2. c6, b7xc6; 3. b6 and white queens.

Figure 14



problem 53
white to move and win

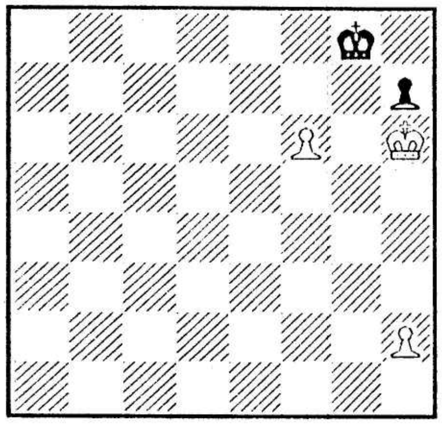
Figure 15



problem 67
white to move and win

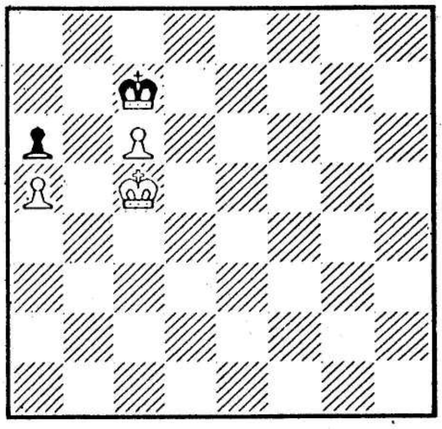
Neither program can solve the above two problems. Peasant cannot carry out the 20+ ply search whereas the co-ordinate squares program fails to account for the changing critical squares. Also, the notion that white can decide how many tempos to consume with a pawn on the second rank is not considered by the existing co-ordinate squares program.

Figure 16



problem 25
white to move and win

Figure 17

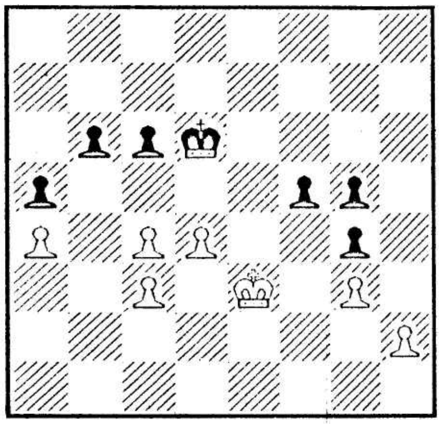


problem 26
white to move and win

Although these problems look very similar, problem 25 is much harder. Neither program can solve 25; both can find the best move in problem 26. There is some question whether Peasant can find the win in problem 26, though. Since, in problem 26, all moves immediately lose the c6 pawn except Kd5, Peasant does find the winning move. To see that black must lose his rook pawn involves a better understanding.

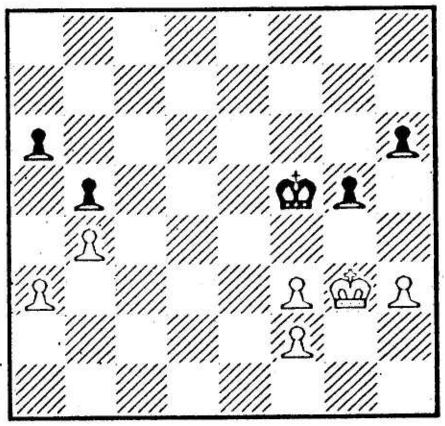
The difficulty with problem 25 is that white has to lock the rook pawns at a favorable time. Wherever the rook pawns are locked, the critical squares will be significantly changed. Also, white must realize that his pawn on h2 can advance either one or two, so white can choose how many tempos he wants.

Figure 18



problem 80
black to move and win

Figure 19

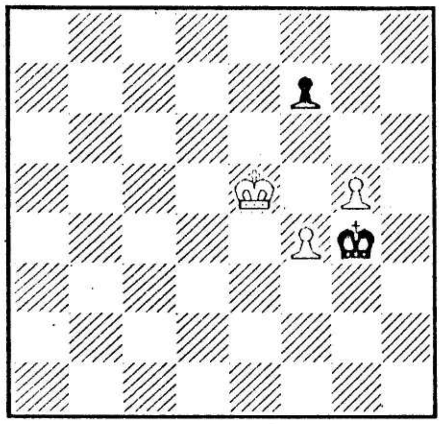


problem 76
white to move and win

Neither program can find the win in either of the above problems. Problem 80 involves a shifting square of two potentially passed pawns, b6 and f5. Unfortunately, since the pawn cluster experts do not know about double pawns, the co-ordinate squares program cannot identify the queenside potentially passed pawn.

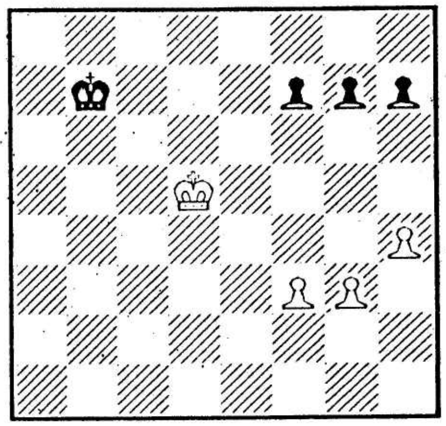
The winning idea in problem 76 is that white can sac the f3 pawn, changing the co-ordinate squares so that white eventually wins a black pawn and the game. Again, this is beyond both programs.

Figure 20



problem 42
white to move and win

Figure 21

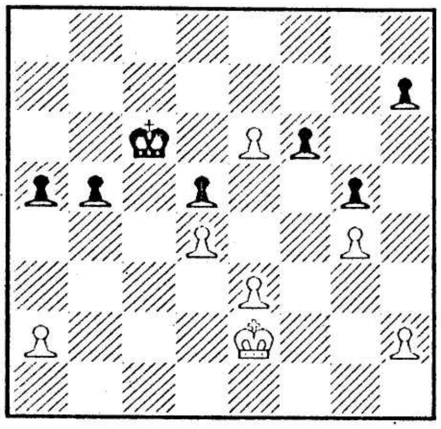


problem 66
white to move and win

The correct move, Ke4, is forced in Problem 42 because everything else loses a pawn immediately. However, seeing the win is somewhat more complicated but not too complicated for the co-ordinate squares program.

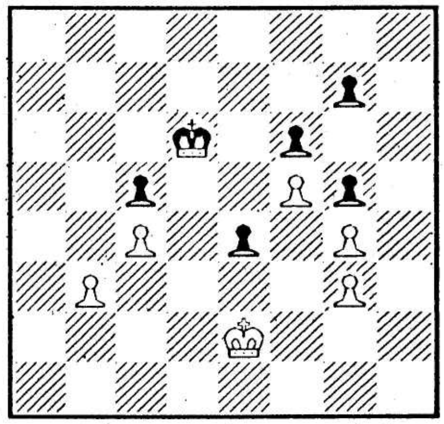
The win in Problem 66 is extremely straight forward, white races after the black pawns. Peasant, though, does have trouble because there are quite a number of useless pawn moves.

Figure 22



problem 90
white to move and win

Figure 23



problem 100A
white to move and win

Problem 90 is a hopeless mess; Fine's solution is a mere 37 plies. A number of necessary concepts are missing from the co-ordinate squares program, most notably, shifting square. There is too much going on in this problem to discuss how a program might solve it.

It is easy to see how white can win the e4 pawn in problem 100A. Unfortunately, co-ordinate squares believes that the game is over once that target is obtained. It happens that it can also see the win after winning the pawn. It really should check that this is indeed the case, though. Peasant isn't capable of seeing the win after winning the e4 pawn.

Conclusion

In conclusion, although this work has shown that co-ordinate squares can solve several endgames otherwise requiring 20-30+ plies of analysis, the question remains, what are the limits to the approach? At first, the algorithm appears to be exponential with the number of potential pawn moves. That is, one way to solve problems with pawn moves is to map the co-ordinate squares for any pawn structure that could result. However, if we think about what the pawn moves mean, then we can do considerably better. For example, a tempo, having one more pawn move than the opponent, is an option to pass. We have shown how to incrementally modify the co-ordinate squares map to account for the tempo. Similarly, the effects of a passed pawn can be considered without building separate co-ordinate squares maps for each square the passed pawn could occupy before it queens. It should be possible to state exactly how pawn moves and king moves are coupled.

At first it was believed that the strategy would break down when more pieces are introduced. Even that is not so clear; pieces, especially slow ones such as knights, in many cases must avoid zugzwang by co-ordinating with each other. In fact, Averbakh has written anther book about knights [Averbakh, *Knight Endgames*].

The key observation that the strategy depends upon is that all these positions have a very limited number of terminal nodes, distinct positions that could result after an arbitrary number of moves. Since there are so few terminal nodes, it is possible to apply a static evaluation function to each one (the first criterion) and then observe how the nodes are connected (triangulation.)

Using the queue improvement to Waltz's algorithm, co-ordinate squares starts with the most constrained nodes and works backwards toward the least constrained. It happens that the constraints tend to be centered near the targets, and that the initial placements of the kings tend to be relatively far from the targets, in the more difficult endgame problems such as Fine #70 (figure 1). Working backward from the target positions to the initial positions, co-ordinate squares is somewhat similar to backward reasoning, just as a depth first tree search from the initial position toward the target positions is forward reasoning. The usefulness of this analogy is in the observation that it is often possible to halve the exponential growth of a search by building backwards from the terminal nodes and forward from the initial position simultaneously. If this could be applied to co-ordinate squares as described above, then the defense sets would not contain a number of unobtainable defenses. For example, in Fine problem #70 (figure 1), a4 is a defense to c4 for black even though there is no way for black to ever reach a4. The section on forward pruning discussed existing heuristics to weed out many unobtainable defenses.

We have designed a system that thinks about the chess domain more as a graph of possible positions than as a tree. Consequently, when the number of positions is small, as it tends to be in king and pawn endgames, the procedure works surprisingly well. In general, as the number of interesting positions grows, the space tends to look more and more like a tree. That is, it becomes less likely that there will be two sensible lines leading to the same position. In a general chess position, co-ordinate squares is not the optimal representation. Within our restricted endgame domain, which has long been a weak point of chess machines, the co-ordinate squares procedure shows great promise. As a generalization of Waltz's filter, this work may have some implications in other domains besides chess endgames.

References

Averbakh, Y. and Maizelis, I., *Pawn Endings*, Chess Digest, Inc., 1974 (English Translation).

Averbakh, Y. and Chekhover, V., *Knight Endings*, B. T. Batsford Ltd., 1977 (English Translation).

Fine, Reuben, *Basic Chess Endings*, David McKay Company, Inc., 1941.

Frey, Peter W. (editor), *Chess Skill in Man and Machine*, Springer-Verlag, 1977.

Perdue, Crispin and Berliner, Hans J., *EG -- A Case Study in Problem Solving with King and Pawn Endings*, IJCAI, 1977.

Tan, S. T., *Representation of Knowledge for Very Simple Pawn Endings in Chess*, University of Edinburgh Memo MIP-R-98, 1972.

Winston, Patrick Henry (editor), *The Psychology of Computer Vision*, McGraw-Hill Book Company, 1975.

Footnotes

1. Peasant, the only chess program designed to play only king and pawn endgames, uses a depth first alpha-beta branching tree strategy. The static evaluator is:

$10 * MAT + 5 * PP - PRO + K_1 + K_2 + R$

MAT ≡ the difference between the number of white pieces and the number of black pieces

PP ≡ the difference between the number of white passed pawns and black passed pawns

PRO ≡ the number of moves the most advanced white pawn must take before promotion minus the number of moves for the most advanced black pawn

K_1 ≡ factor measuring king distance from the pawns; five points deducted for every space the separates the king from the "center of gravity" of the pawns

K_2 ≡ three points if the king has opposition

R ≡ ten points times the rank of each pawn that is passed and cannot be stopped by the defending king.

The forward pruning heuristic eliminates king moves which take the king more than two columns or three rows from the nearest piece and king moves to the edge of the board if there would otherwise be eight legal king moves. The *killer heuristic* [Frey, chapter 3] is used to improve the effectiveness of alpha-beta. Also, moves are sorted so that captures and promotions are first. A position is defined to be a terminal node if one of the following conditions holds:

1. The maximum preset depth is met.

2. One side has one or two pawns and the other has none. (There is a special static evaluator for these case.)

3. There is a queen on the board and the last move was not a promotion.

4. There is a passed pawn which cannot be caught by the enemy king and can outrace all enemy pawns with a move to spare.

5. The depth is equal to that of a node where a win can be guaranteed. (This appears to be a special case of alpha-beta.)

6. The same position has occured previously at the same depth in the tree.

7. Stalemate

8. The position is equivalent to a parent position which occurrs four plies higher in the tree.

9. The winning side allows draw by repetition.

The most serious design error is that rule six is too weak. A better condition is to terminate if the position has been reached previously in the tree search at any depth. Especially in these endgames, this is a very serious error.

2. All chess moves and squares within this paper are given in *Algebraic Notation*, the most common system everywhere except in the United States and England. Each square is referenced with a letter, *a* through *h* and a number, *1* through *8* as shown below:

a8	b8	c8	d8	e8	f8	g8	h8
a7	b7	c7	d7	e7	f7	g7	h7
a6	b6	c6	d6	e6	f6	g6	h6
a5	b5	c5	d5	e5	f5	g5	h5
a4	b4	c4	d4	e4	f4	g4	h4
a3	b3	c3	d3	e3	f3	g3	h3
a2	b2	c2	d2	e2	f2	g2	h2
a1	b1	c1	d1	e1	f1	g1	h1

In this notation, a king moves and captures are denoted by a capital *K* followed by the destination square. Pawn moves are denoted by the destination square. A square followed by an *x* followed by another square denotes a pawn capture.

3. These terms are difficult to define either because they are advanced chess concepts that the author doesn't fully understand or because they have no precise definition.

triangulation - Taking a longer than necessary route toward a target so that the position will recur with other side on the move.

opposition - The state where the two kings are on the same rank or file and there is exactly one square in between.

diagonal opposition - The state where the two kings are on the same diagonal and there is exactly one square in between.

distant opposition - The state where the difference in ranks of the two kings and the difference in files are both even.

co-ordination - (relatively new term) The state where there exists a defensive reply to wherever the attacking king moves.

tempo-ing - A catch-all term to cover arbitrarily complex attempts to repeat the position with the other side on the move. Although triangulation is the most common method, there are other possibilities such as pawn maneuvering.

zugzwang - A state where the side to move has only bad moves but would be better if the other side had the move.

4. Opposition can be twisted by certain pawn formations. Since the opposition is usually an asset, it is understandable why both Fine and Averbakh might overlook the fact that the opposition is not necessary for black to draw.

5. Tempo is yet another difficult term to define. In this case, white has one more pawn move than black so he may in effect, pass once instead of moving his king.

6. The square of a passed pawn is the region where the defending king can catch the pawn before it promotes. It is called a square because, unless there are external obstacles, the region tends to be square. A very good approximation is that the defending king must be closer to either the queening square or the pawn than the attacking king and the defending king must be as close to the queening square as the pawn. The actual implementation considers a few other details such as the fact that pawns can move two squares from the second rank.

7. One way to generate the squares adjacent to a defense set A would be to OR the squares adjacent to each row r_i of A. The squares adjacent to a row could be computed by table lookup. Since there are only 2^8 different ways to label a row of a defense set, the table size would be 256. In this way, we can compute a whole row in parallel. It goes

without saying, that we could generate the squares adjacent to a set completely in parallel using a special purpose processor.

Acknowledgements

I wish to thank Kurt VanLehn for an extremely profitable conversation which led to the queue improvement to Waltz's algorithm. Tim Anderson, Kurt VanLehn, and my especially my father, Russell Church, were among a large number of very supportive readers. And, of course, I mustn't overlook Richard Greenblatt, my adviser.