

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

Kenneth W. Church
NE43-308
MIT
Cambridge, MA. 02139

Abstract: The solution to a number of Artificial Intelligence 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, which is analogous to Waltz' scene analysis algorithm.

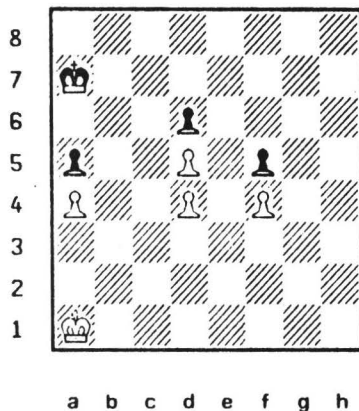
The Co-ordinate Squares theory, which has received considerable attention in the chess literature over the last half century, is implemented using a constraint propagation technique. The constraint is that the defending king must precisely co-ordinate with the attacking king to avoid losing. For any particular square the attacking king occupies, there are only a few possible (co-ordinating) squares the defending king could occupy without transposing into a known lost position. As the attacking king moves from one square to another, the defending king must co-ordinate, move from a co-ordinate of the one square to a co-ordinate of the other. We will give a reduction algorithm which propagates this constraint in order to determine the co-ordinate squares.

Introduction

We will begin by defining a simplified version of king and pawn endgames, which we will call COORD. Although this game can be solved by traditional methods, it is a good example for demonstrating co-ordinate squares. Then we will explore some aspects of king and pawn endgames which are not included in COORD.

COORD is played on a chess board with kings and pawns which move in the usual way. In the initial position, the pawns are all locked against each other so there are no pawn moves. White wins if he captures a black pawn. Black draws if white never wins. The task is to find the next move and prove that it is optimal. Consider the following COORD problem which will be used throughout this paper."

Figure 1



white to move wins; black to move draws

This famous problem has a rich history in the chess literature ([Fine, problem #70], [Averbakh, problem #671], and many others). Computer chess researchers have also studied this problem. Newborn, the author of *Peasant****, estimates that this problem would require 25,000 hours of cpu time [Newborn, page 129]. Chess 4.5 has more recently found the solution with a 26 ply search [Frey2]. Using co-ordinate squares, we can find the solution in about thirty seconds on a Lisp Machine [Weinreb].****

* This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

** All chess moves and squares within this paper are given in *Algebraic Notation*, the most common notation in the world. Each square is referenced with a letter, a through h and a number, 1 through 8 as shown below figure 1. In this notation, 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.

*** *Peasant*, one of the few chess program designed to play only king and pawn endgames, uses a depth first alpha-beta branching tree strategy.

**** More careful analysis of the program performance can be found in [Church]. The co-ordinate squares approach is solution is at least as good as *Peasant*'s on 16 problems from [Fine]. The time cost is shown to be bounded by a constant for king and pawn endgames where pawn moves are irrelevant. Algorithmic analysis of the pawn moving heuristics have not been performed. In practice, the time costs are well within normal chess tournament constraints.

Most chess programs, including Peasant and Chess 4.5, use a depth first search (DFS) which can be hard to contain. DFS might, for example, explore a queen endgame that could follow from a long series of blunders.

Another problem with a DFS is that there are often a large number of paths between any two positions. The DFS algorithm has recently used a transposition table to catch positions that have been previously explored. For solving COORD problems, a transposition table is more expensive than necessary.

A DFS transforms the search space into a tree. A better representation is probably a graph of positions connected by full moves (a half move for each side). In COORD there are only 4096 positions since the pawns are fixed and there are only 64 ways to place each king. Furthermore, the maximum out-degree is 64 since each king can make at most 8 legal moves. This very regular graph is probably more efficient than a more general transposition table.

Given any position, it is theoretically possible to determine who will win. This determination can be made independently of the context, the moves that have been played previously. A position has no memory. A DFS maintains this context on the stack. Perhaps this is a mistake.

To conclude this section, we have motivated that the traditional method, a DFS, may not be optimal for solving COORD problems. In the next section, we will consider an alternative approach which does not suffer from the drawbacks stated above.

An Alternative to DFS

What does determine the value of a position, whether or not white will win? A winning position either is *obviously* winning (inductive basis) or can be forced into another winning position in exactly one move (inductive step). The decisions are local; they could be performed by a parallel machine consisting of 4096 gates comparing each of the 4096 positions with its neighbors. The locality is important because it allows us to resolve the positions in any order without changing the results.

To determine whether a position is obviously winning requires chess domain heuristics which will be discussed later. We will assume for now that an obviously winning position is one where the white king has captured a pawn. Although these domain heuristics strongly influence the results, this paper is more concerned with the inductive reasoning. Demonstrating that this induction is correct requires showing that any unresolved position which leads to a known winning position in exactly one move is itself winning (easy) and that any unresolved position, which doesn't lead to a resolvable winning position in exactly one move, isn't winning (more difficult). Due to space considerations, the demonstration will not be given here.

Even though we may be able to compute the value of every position, we still don't know which move is best. However, we do know that the best move will preserve the value. This provides a very strong constraint for selecting the best move.

We may still need a DFS to choose the best move, but the DFS is over a much smaller search space. The implemented program calculates these constraints, without performing the final DFS to choose a move.

We have motivated an alternative to the traditional DFS. Before describing the algorithm more specifically, we would like to show how this relates to the chess notion of *co-ordinate squares*.

Co-ordinate Squares

We have defined drawing positions to be those positions from which white cannot force a win. In other words, no matter what white does from a drawing position, black has a adequate reply. The black king is *co-ordinating* with the white one. White can move this way or that, left or right, but he cannot out maneuver the black king. The chess technique of co-ordinate squares as advocated by Averbakh and Maizelis [Averbakh] locates many of the co-ordinating squares, using some unspecified procedure, in order to simplify the search process. In this work, we have refined the procedure to an algorithmic process.

The Reduction Algorithm

If black is co-ordinating at one time and a move later he is not, then he must have made a mistake. Similarly, if white is winning and a move later, black is back defending successfully, then white must have missed an opportunity. Co-ordination is invariant. The end result cannot change if both sides play perfectly. Consequently, to decide if white is going to win, all we have to do is decide if he is winning in the initial position.

To decide the value of the initial position, we have to decide the value of the neighboring positions. Chances are that this procedure will request the value of every position. So far this is very similar to a DFS. The difference is that this procedure does not specify the order in which the positions are considered. How can we efficiently classify the 4096 positions into winning and losing? We will use a reduction algorithm. Initially, classify the 4096 positions into one of two categories: either obviously winning for white or unknown. Each snapshot depends upon its neighbors. That is, if an obviously winning position can be forced from an unresolved position, then it too is obviously winning for the attacker (white). The algorithm keeps checking positions with their neighbors until no more unresolved positions can be classified winning. At that point, the unresolved positions are classified co-ordinating because the attacker cannot force a win from them.

The Triangulation Criterion

We use the chess term triangulation loosely for the comparison of a position with its neighbors. This step accounts for arbitrarily complex king maneuvers, some of which chess players call triangulation, opposition, distant opposition, co-ordination, tempo-ing, or zugzwang. The constraint is that the defender must have a reply to any move by the attacker. Assume that the attacker is on a_{attack} and the defender is on a_{defend} . If the attacker can move to b_{attack} , then the defender

must be able to reply by moving to a square b_{defend} which defends b_{attack} . If the defender doesn't have such a reply, then his defense on a_{defend} has been refuted.

Since all defenses of an attacking square (placement of the attacking king) must meet this constraint, we can optimize the comparison procedure with the following reformulation. If the attacker can move from one square to another then all defenses of the former must be a move from a defense of the latter. This formulation of triangulation motivates the notion of a defense set (dset), the set of squares which defend against an attacking square. A implementation is given below using the primitives fetch-dset (to retrieve the dset of an attacking square) and remove (to remove a defense from a dset). The predicate can-move? decides if a player (either white or black) can move from square a to square b when the other king is on square oking.

```
can-move? (player, a, b, oking) =
  adjacent (a, b) & legal (player, b, oking)

triangulate:proc (attacker, aattack, battack)
  A := fetch-dset (attacker, aattack)
  B := fetch-dset (attacker, battack)
  defender := other-side (attacker)
  for all adefend in A do
    if ~∃ bdefend ∈ B
      [can-move? (attacker, aattack, battack, adefend) →
       can-move? (defender, adefend, bdefend, battack)]
      then remove (adefend, A)
```

* The following are common chess terms.

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.

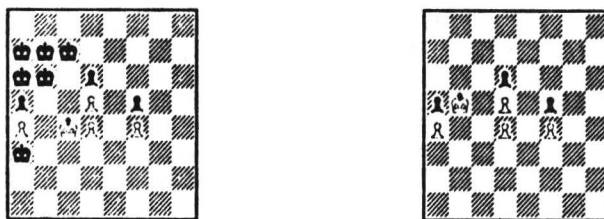
distant opposition - The state where the difference in ranks of the two kings and the difference in files are both even.

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 and would be well off if only he didn't have to move.

Figure 2



The black kings mark the possible defenses (before triangulation); everything else obviously loses since the white king is closer to a black pawn. (This computation is described later in more detail.) Since the white king on c4 (left) is threatening to move to b5 (right), all defenses of c4 ({a7, b7, c7, a6, b6, a3}) must either prevent the threat or respond to it. a6 and b6 prevent the white king from moving to b5. None of the other defenses work because black cannot respond to the threat by moving from one of them to a defense of b5. At this point $A = \{a6, b6\}$.

These reductions tend to propagate. Now that a number of defenses of c4 have been eliminated, it is possible that one of c4's neighbors might require further reduction. This process is defined to terminate when no more reductions are possible. The propagation effect explains why a6 is eventually removed from the defense set of c4. If black were on a6, white could beat black to the f5 pawn. 1. Kd3, Kb6; 2. Ke3, Kc7; 3. Kf3, Kd7; 4. Kg3, Ke7; 5. Kh4, Kf6; 6. Kh5, Kf7; 7. Kg5, etc. The algorithm determines this by triangulating the dset of g5 with that of h5 and then the dset of h5 with that of f6 and so on. The propagation continues until the map has settled.

A Solution to Figure 1

Since the order of triangulating will not change the final results as long as we keep triangulating until the propagations have settled, we will adopt the following non-optimal implementation for expository purposes only:

Initialization: assign the first guess dsets (using chess domain heuristics):

Do until no dset changes
triangulate each dset with all its neighbors

The first guess dsets are shown below. For the sake of brevity, we will show only the more interesting sets.

```
b5 {}
c4 {a6, b6, a7, b7, c7} (ignoring defenses like a3)
b3 {a6, b6, a7, b7, c7, d7, a8, b8, c8, d8}
c3 {a6, b6, a7, b7, c7, d7, a8, b8, c8, d8}
d3 {a6, b6, a7, b7, c7, d7, a8, b8, c8, d8}
e3 {b6, b7, c7, d7, e7, b8, c8, d8, e8}
f3 {f6, c7, d7, e7, f7, c8, d8, e8, f8}
g3 {f6, g6, d7, e7, f7, g7, d8, e8, f8, g8}
h4 {f6, g6, h6, e7, f7, g7, h7, e8, f8, g8, h8}
g5 {}
h5 {f6, e7, f7, g7, h7}
```

h6 {f6, e7, f7, g7, h7}
 b2 {a6, b6, a7, b7, c7, d7, e7, a8, b8, c8, d8, e8}
 c2 {a6, b6, a7, b7, c7, d7, e7, a8, b8, c8, d8, e8}
 d2 {a6, b6, a7, b7, c7, d7, e7, a8, b8, c8, d8, e8}
 a1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}
 b1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}
 c1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}
 d1 {a6, b6, a7, b7, c7, d7, e7, f7, a8, b8, c8, d8, e8, f8}

Triangulate the above sets with their neighbors.

Iteration 1

c4 {a6, b6} (constrained by the dset of b5)
 h4 {f6, g6, h6} (constrained by the dset of g5)
 h5 {f6} (constrained by the dset of g5)
 h6 {f6} (constrained by the dset of g5)
 all others unchanged

On the next iteration, by triangulating all the dsets again, the constraints propagate another square away from the targets.

Iteration 2

b3 {a6, b6, a7, b7, c7}
 c3 {a6, b6, a7, b7, c7}
 d3 {a6, b6, a7, b7, c7}
 h4 {g6, h6}
 h5 {}
 h6 {}
 g3 {g6, h6, e7, f7, g7, h7}

This process continues until no more defenses can be removed. These iterations have been deleted for brevity.

Summary

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}

White wins if he has the move because black's king on a7 does not defend a1. He must play moves which prevent black from co-ordinating. One winning line is: 1. Kb1, Kb7; 2. Kc1, Kc7; 3. Kd1, Kd7; 4. Kc2, Kc7; 5. Kd3 and black has to move. If he plays Kb8, Kc8, Kd8, or Kd7 white wins the a5 pawn with Kc4. On any other black move, white wins the f5 pawn with Ke3. The algorithm does not tell us how to win; it only decides which moves preserve the win. In this case, white has only one move (1. Kb1) which preserves the win. Black can reply to 1. Ka2 or 1. Kb2 by moving to a co-ordinate, {b7, b8} and {a8}, respectively. From these co-ordinating positions, no matter where white moves, black will be able to move to another co-ordinating position.

In the same position with black to move, black can co-ordinate by moving to either b7 or b8, since both defend a1. Neither Fine nor Averbakh suggest Kb8 because that defense is somewhat "anti-theoretical" probably because it gives away a twisted distant opposition*. Kb8 may look unusual because it unnecessarily moves away from the action.

The Order of Triangulating Dsets

Even though the order will not change the final results, it can influence the running time. For best efficiency, 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. Initially all the sets are on the queue. In each iteration, pull a set off the queue and triangulate it with its neighbors. If triangulation should remove any elements from a neighboring set, then check for the propagations by re-queuing those neighboring sets.

How do we decide if one set is more constrained than another? One answer is to employ some domain restricted knowledge. For example, we might use a heuristic which suggests that attacking squares closer to pawns tend to have more constrained defense sets. However, in this work, we have chosen a much simpler and domain independent ordering heuristic. The smaller dsets are more constrained than the larger ones. This completes the description of triangulation.

The Triangulation Criterion Viewed as a Waltz Filter

It is useful to compare this approach to previous work in other areas of AI. In this section, we will show that this reduction technique is very similar to 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.

* 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.

** See [Mackworth].

The triangulation procedure is 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). Using the queue mechanism discussed previously, it is possible to improve the average cost. In Waltz's filter, since the propagation possibility is checked immediately after a set is reduced, it is unlikely that they will be checked in the best order. (The order wasn't as important to Waltz because the line drawing world has only 4 possible labels whereas the chess endgame world has 64.) Another advantage of the queue is that it can avoid considering some vertices multiple times. The Waltz filter will check a set for propagations that it will have to check later anyways. The queue mechanism avoids this by preventing a set from appearing twice on the queue. It is very difficult, though, to see exactly how good the queue is.

Finding Possible Defenses

In this section we will discuss the chess domain heuristics involved in deciding the "obvious" cases. These are only heuristics; although the more complete heuristics discussed in [Church] have no known limitations for COORD examples, they do not solve all king and pawn endgames. That work illustrates the limitations in more detail.

One criterion which will work for COORD, but will be difficult to extend to practical chess endgames, is the following: *Target positions (the white king occupies the same square as a black pawn did) are obviously winning. All others might defend.* In this subsection, we will define a more useful criterion.

For now we will assume that the defender will lose unless his king is as close to every target, black pawn, as the attacking king. This definition uses the notion of distance. White's distance between two points is the number of white king moves it would take to travel the distance. Black's distance measures black king moves. The two are not the same because a king cannot move on top of a friendly pawn or into an enemy pawn's zone of control. There is an algorithm for computing the distance from one point to all others.

Closeness

A player's king (on square *a*) is closer to the target *t* than the other player's king (on *d*) if his distance to the target is less than the other player's.

```
closer (player, a, d, t) =
  distance (player, a, t) <
  distance (other-side (player), d, t)
```

Defense Set

A square *d* possibly defends an attacking square *a* if it is legal for the the defender and close enough to the targets.

```
dset (attacker, a) =
  {d | legal (other-side (attacker), d, a) &
    ∀ t ∈ 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 five from *f5*. According to this first criterion, black can defend *c4* from *a7*, *b7*, *c7*, *a6*, *b6*, or *a3*. (See Figure 2 shown above.) 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 have seen that triangulation will remove many of these defenses.

Extended COORD

As we have said, COORD is simpler than chess king and pawn endgames. In this section, we will explore methods to extend co-ordinate squares. One problem with COORD is that black cannot win. One solution is to do the calculation twice, once where white is the attacker (as in COORD) and once where black is the attacker. The game is drawn if both sides are co-ordinating with each other. It should be impossible for neither side to be co-ordinating with the other. If only one side is co-ordinating with the other, then he is winning. There are probably more attractive solutions.

There are quite a few other differences between COORD and chess king and pawn endgames which must be considered. We have two basic strategies to extend co-ordinate squares. The first, which is untested, increases the search space. Co-ordinate squares is very good in manipulating the kings in one pawn configuration. However, there are ways to jump into another pawn configuration, by either capturing or moving a pawn. Once a player jumps into another pawn configuration (hyper-space), he can never return because there is no way to move a pawn backwards or to bring back a captured pawn from the dead. The graph of hyper-spaces is therefore acyclic. We could solve each hyper-space one at a time, starting with the leaves until we have solved the root space (the initial position.) The problem with this idea is that the number of hyper-spaces can be very large, one for each possible pawn configuration. Five pawns on the second rank would generate 5^7 spaces. There must be heuristics to reduce the number of hyper-spaces.

The other approach augments the existing search with the necessary chess heuristics. We [Church] have solved a large class of king and pawn endgames with this approach although there are problematic endgames for which the heuristics do not apply. The optimal solution is probably a compromise between the two approaches. It would be worthwhile to experiment with the hyper-space approach.

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 can show how to incrementally modify the co-ordinate squares map to account for the tempo [Church]. 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 another 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 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. [Church] describes some heuristics for quickly removing such useless defenses.

We have designed a system that thinks about the chess domain more as a regular 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. 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).
- Church, Kenneth W., *Co-ordinate Squares: a Solution to Many Chess Pawn Endgames*, Bachelor's Thesis at MIT, 1978.
- Fine, Reuben, *Basic Chess Endings*, David McKay Company, Inc., 1941.
- Frey, Peter W. (editor), *Chess Skill in Man and Machine*, Springer-Verlag, 1977.
- Frey, Peter W., personal communications, 1978.
- Mackworth, Alan K., *Consistency in Networks of Relations*, AI Journal, February 1977.
- Newborn, Monroe, *PEASANT: An endgame program for kings and pawns*, chapter 5 of *Chess Skill in Man and Machine*.
- 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.
- Weinreb, D. and Moon, D., *Lisp Machine Manual*, MIT, 1978.
- Winston, Patrick Henry (editor), *The Psychology of Computer Vision*, McGraw-Hill Book Company, 1975.

Acknowledgments

Tim Anderson, Kurt VanLehn, and my especially my father, Russell Church, were among a large number of very supportive readers. Hans Berliner has provided much encouragement. And, of course, I mustn't overlook Richard Greenblatt, my thesis adviser and friend.