Selective Extensions in Game-Tree Search †

Chun Ye
T.A. Marsland

Computing Science Department
University of Alberta
Edmonton
CANADA T6G 2H1
email: {chunye,tony}@cs.ualberta.ca

ABSTRACT

Although most of today's chess playing programs still adopt a brute-force approach in their search region, much has been done on search extensions to make the effort spent more worthwhile. In this paper, we discuss some successful search extension heuristics in the domain of Chinese Chess, a game that bears much resemblance to chess. We restrict our experiments to the following: knowledge search extensions, singular extensions, null move search (both in the brute-force and the quiescence search phase) and futility cutoffs. These heuristics have been implemented in Abyss, a Chinese Chess program participating in the 3rd Computer Olympiad. From the algorithmic point of view, since Chinese Chess differs most from chess in its repetition rules, some discussion is also devoted to that matter.

1. Introduction

Most of today's chess playing programs still adopt a brute-force framework for their search region, yet almost all of them employ selective extensions on particular moves during the search to make the effort spent more worthwhile. Apart from the complicated repetition rules, Chinese Chess is similar in nature to chess (perhaps more tactical since in Chinese Chess no promotion of pawns is possible, so that all games have to be won through a successful attack on the opponent's king). It is natural that equivalent extension heuristics should exist for Chinese Chess but whether they are effective and whether the different evaluation functions, promotions and repetition rules affect their properties is the motivation of this paper.

† This paper is based on the ongoing work by Chun Ye as part of his M. Sc. research at the University of Alberta, and is partially supported by the Natural Sciences and Engineering Research Council of Canada under Grant A0792 held by the second author.

To narrow the focus of our discussion, we restrict ourselves to the following heuristics drawn from computer chess; first, extensions using domain specific knowledge; second, singular extensions where information gathered from the search itself is used to decide whether to search deeper; third, null move search (both in the brute-force and quiescence search phase) and last, futility cutoffs, a means of forward pruning some seemingly futile moves with little risk. Some space is also devoted to the repetition rule of Chinese Chess, since it differs in a major way from repetition handling in chess.

2. Search Extensions on Forcing Moves

With today's faster hardware and enhanced search algorithms, it is possible for the best chess playing programs to search to a formidable depth (say 9-ply or even deeper) during most of the middle game. Even for those programs, moves beyond the game tree horizon may be neglected. To alleviate this problem, a more promising approach is adopted viz., *selective extensions*, increasing the search by an extra ply (plies) when certain criteria are met. For instance, chess playing programs will usually extend the search with an extra ply when the side to move is in check, since checking usually consists of a serious threat. The safety of a deeper search is worth the extra-cost, which isn't high since the number of replies to a checking move is small. This is one example of using domain-specific knowledge to extend the search depth. Other approaches include extending on recaptures [Ebeling 1987, p. 101-2], pawn moves to the 6th and 7th rank in chess [Kaindl 1982, Anantharaman *et al*. 1988], moves near the territory of the opponent's king [Anantharaman 1991], strictly forced moves (say if one side has only one legal move) [Uiterwijk 1991] and certain piece evading moves to bring a piece out of the opponent's attack (an *ad hoc* heuristic tried in *Abyss*). The latter two have not yet been adequately explored in the computer chess literature, but both have a sound foundation.

Here we give descriptions for these search extension heuristics, some experimental results, and share our experience in implementing them in the *Abyss* Chinese Chess program. The search extension heuristics experimented with include: *check evasion*, *recaptures*, *futility-cutoff*, and *null move search*. Results from eleven combinations of the four heuristics are presented, as tested against 50 Chinese Chess middle games drawn from a standard work [Tu 1985]. These results are summarized in Table 1, through data from a series of experiments defined in Table 2. In those tables the success rate is the fraction of the 50 positions correctly solved. A more accurate measure (considering only those moves that not only meet the solutions but also provide correct principal variations) will be used in the M.Sc. thesis in preparation [Ye 1992].

Table 1: Experiment results for different extension heuristics (over 50 positions)

Exp. No.	depth = 3			depth = 4			depth = 5		
	node count	success rate	node ratio	node count	success rate	node ratio	node count	success rate	node ratio
1	202898	26%	0%	1607058	30%	0%	5793297	32%	0%
2	251769	32%	24%	1572444	36%	-2%	7333702	52%	27%
3	207583	26%	2%	2067686	30%	29%	6598971	32%	14%
4	259419	32%	28%	2005139	38%	25%	8050061	54%	39%
5	163352	24%	-19%	1143982	28%	-29%	3147045	30%	-46%
6	213529	32%	5%	1065341	34%	-34%	4602454	50%	-21%
7	165255	24%	-19%	1217646	28%	-24%	3568519	30%	-38%
8	222135	32%	9%	1115973	36%	-31%	5144946	50%	-11%
9	175702	26%	-13%	1125369	32%	-30%	4085124	30%	-29%
10	219115	34%	8%	1426952	38%	-11%	4735374	54%	-18%
11	195930	34%	-3%	856910	36%	-47%	3319976	50%	-43%

Table 2: Flag settings for different experiments

Exp. No.	Configuration
1	– check_evasion – recapture – futility_cutoff – null_move
2	+ check_evasion – recapture – futility_cutoff – null_move
3	– check_evasion + recapture – futility_cutoff – null_move
4	+ check_evasion + recapture – futility_cutoff – null_move
5	– check_evasion – recapture + futility_cutoff – null_move
6	+ check_evasion – recapture + futility_cutoff – null_move
7	– check_evasion + recapture + futility_cutoff – null_move
8	+ check_evasion + recapture + futility_cutoff – null_move
9	– check_evasion – recapture – futility_cutoff + null_move
10	+ check_evasion + recapture – futility_cutoff + null_move
11	+ check_evasion + recapture + futility_cutoff + null_move

2.1. Search Extensions using Domain Specific Knowledge

2.1.1. Check Evasion

Because of its simplicity and efficiency, *check evasion* is perhaps the most commonly found feature in chess programs. A checking move usually forms a major threat and is a forcing move, therefore a one ply deeper search might reveal some tactics that are beyond the original horizon. The situation is of course substantially the same in Chinese Chess, so one can expect a similar benefit from adding such a heuristic.

Our results confirm this assumption, since although adding the *check evasion* heuristic caused the nodes searched to be increased (by an average of 16%; Experiment 1 and Experiment 2, Table 1), the cost was acceptable because more correct moves were successfully found (an average of 33% more; Experiment 1 and Experiment 2, Table 1).

2.1.2. Recaptures

Capturing is the essence of tactics in chess (and Chinese Chess), and a capture search [Bettadapur and Marsland 1988] forms the kernel part of quiescence search. In the exhaustive search region, some captures are more or less forced, e.g., *recaptures*, as defined by Carl Ebeling [1987, p. 101-2]. Therefore it might be worthwhile to extend one more ply on recaptures with the hope that some deep tactics can be revealed.

Note that extending all recaptures could be expensive since a capture doesn't restrict the move choices by the opponent, therefore care is needed to avoid a search explosion. In *Abyss*, we adopt the same rule as in *Hitech* [Ebeling 1987, Berliner 1989]; Only recaptures that bring the material merit into a window of the initial root value are considered.

Our experiments show that adding *recaptures* solely doesn't improve the success rate (Experiment 1 and Experiment 3, Table 1), it may however improve the play of the program. Also Experiments 2 and 4 show that improvements (2 percentage points) are possible when *recaptures* is combined with the *check evasion* heuristic.

2.1.3. Other Forcing Moves

There are other moves in Chinese Chess which may be considered 'forcing' and are worth searching with an extra ply. We will discuss three possibilities here, none of which are included in the current version of *Abyss*.

During the 3rd Computer Olympiad, Jos Uiterwijk (author of the chess playing program *Touch*) mentioned that he had implemented another heuristic: where a node has only one legal move it is extended an extra ply [Uiterwijk 1991]. If there is only one legal move in a position, it is of course 'forced' and a less turbulent and more reliable value may be returned after searching by an extra ply. Such a heuristic is especially useful in situations where one side can make a move which leads to a decisive advantage (like a mate threat) but the opponent can 'thwart' this threat by making some delaying moves like checks. By disregarding these moves, it is possible that we can avoid being 'fooled' into missing the threat. Notice that when there is only one legal move in a particular position, this usually means the king of the side to move is under check, therefore two extra plies should be extended when combining with the *check evasion* heuristic.

Another possibility is to extend the search on moves that bring to safety the piece under attack. This may be viewed as a generalized case of the *check evasion* heuristic, since otherwise the piece under attack will be captured by the opponent in the next move, resulting in a material deficit. Moving the piece to safety (to a square where it is no longer under opponent's attack or is protected by pieces of its own side) is somehow forced.

The other reason of using this *ad hoc* extending heuristic in the search algorithm in *Abyss* stems from consideration of the repetition rule of Chinese Chess. *Null moves* can usually be used to detect threats [Anantharaman 1991], but not all threats identified by the *null move* follow the rules of Chinese Chess, and the expense of such a detection is high. Therefore, some simplification to restrict the type of moves which are considered 'threat' is made in *Abyss* (see Section 5 for more details about the repetition check algorithm). As a by-product of detecting a *piece evading* move, we gain knowledge about how to distinguish a legal repetition from an illegal one.

In *Abyss*, a simplified version of this heuristic is included; it only considers moving a major piece out of attack, since including all types of pieces proved to be too expensive.

There is still another type of move that is worth considering, i.e., moving a piece near the enemy's king [Anantharaman 1991]. In Chinese Chess, the king is more vulnerable to attack than in chess. This is because the king in Chinese Chess is confined to only 9 squares (called the *palace*), and no extra protection is possible through a pawn promotion. So the chance is higher here that a game ends through a direct attack upon the opponent's king. In fact, in Chinese Chess there is an adage which says "three pieces besides wins the game"; Chinese Chess players consider positions with three pieces near the opponent's palace as winning, and will choose plans to aim for or avoid this combination. This shows how strong these moves (toward the opponent's king) can be, and therefore it is certainly worth extending the search when these positions occur. However, it might be more efficient to consider only moves that bring the third piece near the territory of the opponent's king.

The above mentioned search extension heuristics are not yet included in the current version of *Abyss* and remain part of our ongoing work.

2.2. Singular Extensions

Using domain-specific knowledge to extend search depth is of course beneficial. Although experiments [Hyatt *et al*. 1990, Ye 1992] show that a n-ply search enabled with extension heuristics is no better than a (n+2)-ply exhaustive search, it outperforms (n+1)-ply exhaustive search in two ways: first it searches fewer nodes (time) and second the move quality is better.

Nevertheless, there are two problems when using domain-specific knowledge to extend the search. As Anantharaman *et al*. [1988] state:

> "First, it is difficult to provide enough knowledge to cover all or most of the interesting cases. Second, the knowledge is usually based only on the static

features of the moves without taking into account the dynamics of the position, and the search extensions based on such knowledge may be grossly irrelevant and wasteful."
A more powerful search extension heuristic called *Singular Extensions* was presented by Anantharaman *et al*. [1988], and proved to be a great success in the chess playing program *Deep Thought*. The idea of *Singular Extensions* is to use information gathered in the search itself to extend the search whenever one move is significantly better than the sibling moves.

There are two types of singular moves which are considered during the search; they are *Singular PV* moves and *Singular Fail-high* moves, which are defined elsewhere [Anantharaman *et al*. 1988]. Both are considered in the current version of the *Abyss* program. However, although the *Singular Fail-high* heuristic was built according to the definition, some simplifications were made to the *Singular PV* heuristic implementation, because of limited programming and testing time before its use at the 3rd Computer Olympiad.

Normally, a re-search is required when the value of a *PV Singular* move drops below α after it is extended with an extra ply. There are many implementation complications related to handling this situation [Anantharaman *et al*. 1988, Anantharaman 1991]. *Abyss* simplifies the treatment by only searching the remaining moves to a nominal depth (without considering any singular extensions) when a move is found to be *PV Singular* but drops its value afterwards when it is extended. Of course, such a treatment is purely because of the time available for programming.

Complete results for the *Singular Extensions* experiments are not yet available. However, both self-playing and playing against a commercial Chinese Chess program (*Xian* [Jacobs 1989]) bring some promising results, but a more complete discussion of this heuristic is left to the upcoming thesis [Ye 1992].

3. Null Move Heuristic

Abyss employs two different heuristics when using the *Null Move* to decide whether to search deeper. First, *Abyss* uses the *Null Move* heuristic, as described by Goetsch and Campbell [1990], in its brute-force region; and second, *Abyss* carries out a *Null Move Quiescence Search*, as proposed by Don Beal [1989], before it starts a capture search [Bettadapur and Marsland 1988].

The *Null Move* heuristic [Goetsch and Campbell 1990] is a means of improving search speed with only little risk. *Abyss* tries a null move search in the internal nodes with a depth reduction of 1 ply before it starts searching legal moves. If the value returned is greater than β, this value is accepted as a true cutoff; otherwise, this value is used to improve the α bound.

The *Null Move* heuristic proves to be successful in *Abyss*. The savings are great; An average node reduction of 24% (Experiment 1 and Experiment 9, Table 1) is achieved by the *Null Move* heuristic alone. When combined with other search extension heuristics, like *check evasion* and *recaptures*, not only does the null move heuristic reduce node expansions, but it does not hurt the average success rate either (compare Experiment 1 with Experiments 4 and 10, Table 1). Actually, *Abyss* disallows the null move search in some critical lines when it knows that the side to move is under threat; this could be a check or certain capture moves. Also, the null move search is not used near frontier nodes [Marsland 1987].

The other heuristic using the null move concept is the *Null Move Quiescence Search* [Beal 1989]. *Abyss* separates its search into three phases: brute-force, null move quiescence and capture search. The purpose of adding a *Null Move Quiescence Search* layer is to find some general threats by the least cost. Our initial experiments show that the average branching factor of the null move quiescence search is about 2, and decreases as the depth of the search deepens, as Table 3 shows.

Table 3: Branching factor of null move quiescence search (1 ply exhaustive)

Depth	2	3	4	5	6
Branching Factor	3.26	2.88	2.38	2.24	1.85

One problem related to implementing the *Null Move Quiescence Search* is how many plies will be feasible. The algorithm is self-terminating but for computers slower than a Sun SPARC, it is probably impractical to search without a depth limit. The depth that *Abyss* uses is 4 ply, since in Chinese Chess (and also in chess) most of the threats can be detected by a 4-ply brute-force search. These threats should also be found by using the *Null Move Quiescence Search*, provided they are purely tactical. However, we lack complete data to confirm that a depth of 4 ply is optimal for micro-computers.

4. Gamma Algorithm, Razoring and Futility Cutoff

The final search extension heuristic that is considered in *Abyss* is the *futility-cutoff*. This heuristic actually provides forward pruning, instead of extending the search. It is however another means of performing a selective search and is therefore included here.

The idea of futility cutoff isn't new, Newborn [1975, p. 177-8] presented a method which he called the Gamma algorithm. The idea is to end the search if the material merit of the current node in the search tree is worse than that of the node making the best move found so far. A similar idea is used in Chess 4.5 too [Slate and Atkin 1983]. There is also the pruning technique by Kent and Birmingham [1977] called *razoring*, which tries to terminate the search if the merit of the current node exceeds the β bound. All these heuristics are generally applicable and closely resemble Newborn's Gamma algorithm, but using them involves some risk. A safer variation is the heuristic which Jonathan Schaeffer calls the *futility-cutoff* [Schaeffer 1986, p. 33-4]. The main differences are: First, use of a *futility-cutoff* is restricted to the layer before the frontier nodes in the search tree. Second, material merit is used to decide whether to stop the search or not, but here the total value of the material merit and the maximum positional value is used. Third, the search doesn't stop when such a criterion is met, instead, it uses this information to forward prune most of the moves and only considers those which bring the material merit into the current window; this consists of all checking moves and some of the captures. In other words, the *futility-cutoff* is a low risk transformation of nodes near the frontier into tip nodes when certain criteria are met. Although this heuristic is often mentioned, only Schaeffer [1986] provides quantitative data to show its effectiveness. Here we offer results based on data gathered from the search of 50 Chinese Chess middle game positions.

Table 4 shows that the *futility-cutoff* heuristic gives as great node count savings (an average of 32%) as the *Null Move* heuristic, while retaining a small move choice error (up to 4%). Combining both heuristics result in even better savings (an average of 47%; see Experiment 4 and Experiment 11, Table 1) without worsening the moves selected, although it is hard to say which heuristic gives more savings (Experiment 8 and Experiment 10, Table 1). However, using *futility-cutoff* proves to be a little riskier and the right move success rate deteriorates when the depth goes deeper (2% at depth 4 and 4% at depth 5, Table 4). Nevertheless, because of its great savings, especially when combing with the *Null Move* heuristic, it is still advisable to include this heuristic in the search.

Table 4: Node savings and success errors when using a futility-cutoff

Comparisons	depth = 3		depth = 4		depth = 5	
	savings	errors	savings	errors	savings	errors
Exp. 1 and Exp. 5	19%	4%	29%	4%	46%	2%
Exp. 2 and Exp. 6	15%	0%	32%	2%	37%	2%
Exp. 3 and Exp. 7	20%	2%	41%	2%	46%	2%
Exp. 4 and Exp. 8	14%	0%	44%	2%	36%	4%
Exp. 10 and Exp. 11	11%	0%	40%	2%	30%	4%

5. Impact of the Chinese Chess Repetition Rule

Although similar to chess, Chinese Chess differs significantly in its repetition rule. For example, repetition check is considered a draw in chess, but such a repetition is not allowed in Chinese Chess. In general, the rules of Chinese Chess disallow the use of certain repetitions after a threat move (even so there are exceptions). Three types of moves are considered as threats; first, checking moves; second, moves that threaten to win material; and last, moves that threaten to mate. However, the rule allows certain repetitions via a ‘threat’, provided the current position is a repetition and is reached by a threat move as well (again there are exceptions to this).

Since the repetition rule of Chinese Chess is so complicated, none of the current Chinese Chess programs can claim that they handle all situations correctly. Tsao Kuo-Ming *et al*. [1990] have proposed a means for their program *Chess Master* to handle most of the commonly occurring situations. The commercial Chinese Chess program *Xian* [Jacobs 1989] guarantees never to make an illegal repetitive move, but still lacks the knowledge to handle cases when a repetition would be legal; and in some cases it allows the opponent to make an illegal repetitive move. Another program, *Surprise*, a participant at the 3rd Computer Olympiad, allows certain check repetitions (which are illegal), when it finds that all other alternatives are significantly ‘worse’ (as evidenced by *Surprise*’s play during the 3rd Computer Olympiad).

In *Abyss*, we tried a more generalized repetition detection algorithm, differentiating between detection in the root and during interior nodes. The scenario behind this is to use a more strict rule for the root node but be ‘generous’ to internal nodes.

For internal nodes, some approximation is made and only check repetition and some simple piece-winning threats are considered. Two positions are considered identical if their transposition table *locks* [Zobrist 1970, Marsland 1987] are the same. A stack (sequential table) is used to store all such *locks* from the first move in each game, no count of the number of repetitions is kept. If a repetition under such a definition is detected, we determine not only whether the move reaching this position is a check, or a threat to win a lone piece, but also that the previous move is not such a simple threat; If the preceding move was not a threat, then a threatening move leading to a repeated position is assigned an *illegal* score (almost as poor as a *mate* score). Any other combination of moves to a repeated position is given a *draw* score and in both cases there is no further search. The reason for using an *illegal* score is because the definition of repetition here is *approximated*, and so we might miss the only possible defense if an *illegal* score is not better than a *mate* score.

Nevertheless, at the root node an illegal move will be disallowed, so a more strict repetition check algorithm is adopted. For each move being considered at the root, we check backwards to see if this position is being repeated for the third time; if so, and the move that reaches this position falls in the category of *threat* (as defined in Section 5.1), we backup to see if the previous position is also a repetition (not necessarily for the third time) and whether the move reaching that position is also a *threat*. If the test of the second position fails (the position is not repeated or the move to it is not a threat), we assign the value of the current move as *forbidden*, a score worse than *mate*; in all other cases when a three-fold repetition is detected, a *draw* score is assigned. We haven’t yet had an opportunity to test the program with a large suite of real problems, so it is too early to say how well the algorithm we adopted in *Abyss* handles the more difficult repetition conditions in Chinese Chess.

5.1. The Definition of Threat

In *Abyss* a *threat* is defined as a:

> *Checking threat*: If a move delivers check, it is a threat. This is the simple case.
>
> *Mate threat*: If after one side has made a move, the opponent can be mated by a series of checking moves, the first move is considered to be a *mate threat*. In *Abyss*, an approximation is adopted. We do a search to a depth of 3, considering only checking moves and replies to check, and assume that there is a mate if a mate score is returned for this search. We believe that the chance of having a mate-in-n ($n > 3$) *threat* is too rare to be worth including in the repetition detection algorithm.
>
> *Piece-winning threat*: Again some simplifications are made, and we only consider those moves that threaten to win an unprotected piece (passed pawns and all minor and major pieces in Chinese Chess).
> The expense of *threat* detection isn't as large as it seems, because the above operations are only carried out when a third-time-repetition is seen, and that test is only done once during the search.

Further work is required to consider more backward positions when a repetition is found. At the moment we can handle some difficult repetition situations, e.g., two threats over two threats (a draw) or two threats over one threat (a loss). Also, because of time limits, search to some predefined fixed depth might be required to detect whether a side has a *piece-winning threat*. By restricting the moves to only captures, checking moves and replies to checks (an extended quiescence search), we can do a search after making a null move (in this case making two consecutive moves for the side causing the repetition). If the value returned exceeds a certain amount (the *threat margin*), the first move can be thought of as a *threat* and can be treated accordingly.

6. Future Work

Selective Extensions has proved to be a promising approach to adding selectivity in today's (Chinese) chess-playing programs using brute-force search algorithms. In this paper, we discussed some extension heuristics adopted in the Chinese Chess playing program *Abyss*. Apart from the experiments described here, there are some other things which are worth considering for future work.

First, before making further comparisons, we must implement a full version of the *Singular Extensions* heuristic, to determine its full effectiveness.

Second, we must combine different extension heuristics. *Null Move Quiescence Search* is an efficient means in detecting threats. The use of *Singular Extensions* is another way of revealing deep combinations, but which of the two is more cost-effective? Further, is there a hybrid which will be better than any single heuristic?

Third, we must find an optimal set of these heuristics for micro-computers. Most of the described heuristics were tested on today's fastest hardware. It is possible that not all of them will be equally effective for micro-computers. Therefore, it is highly desirable that an optimal combination be found and some of the heuristics be excluded.

We expect that at least some of these questions will be answered after the experimental work for the current M.Sc. thesis research is complete.

7. Acknowledgment

We thank Alan Sharpe and Bin Zhang for helping with the testing of *Abyss*, and Dr. Nick Jacobs for making the code of his *Xian* program available. We also recognize the value of Don Beal's discussions about null move heuristics, during his three-month stay as a Visiting Professor.

Bibliography

T. Anantharaman, M. Campbell and F. Hsu, "Singular Extensions: Adding Selectivity to Brute-force Searching," *Int. Comp. Chess Assoc. J.*, 11(4), 1988, pp. 135-143.

T. Anantharaman, "Extension Heuristics," *Int. Comp. Chess Assoc. J.*, 14(2), 1991, pp. 47-65.

D.F. Beal, "Experiments with the Null Move," in *Advances in Computer Chess 5*, D. Beal (ed.), North-Holland, 1989, pp. 65-79.

H. Berliner, "Some Innovations Introduced by Hitech," in *Advances in Computer Chess 5*, D. Beal (ed.), North-Holland, 1989, pp. 283-293.

P. Bettadapur and T.A. Marsland, "Accuracy and Savings in Depth-Limited Capture Search," *Int. J. Man Machine Studies*, 29(5), 1988, pp. 497-502.

J.A. Birmingham and P. Kent, "Tree-searching and Tree-pruning Techniques," in *Advances in Computer Chess 1*, M. Clarke (ed.), Edinburgh Univ. Press, Edinburgh, 1977, pp. 89-107.

C. Ebeling, *All the Right Moves: a VLSI Architecture for Chess*, MIT Press, 1987.

G. Goetsch and M. Campbell, "Experiments with the Null Move Heuristic in Chess," in *Computers, Chess, and Cognition*, T.A. Marsland and J. Schaeffer (eds.), Springer-Verlag, 1990, pp. 159-168.

R.M. Hyatt, A.E. Gower and H.L. Nelson, "Cray Blitz," in *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer (eds.), Springer-Verlag, 1990, pp. 110-130.

N.J.D. Jacobs, "Xian, a Chinese Chess Program," in *Heuristic Programming in Artificial Intelligence*, D. Levy and D. Beal (eds.), Ellis Horwood, London, 1989, pp. 104-112.

H. Kaindl, "Quiescence Search in Computer Chess," *SIGART Newsletter*, Vol. 80, 1982, pp. 124-131.

T.A. Marsland, "Computer Chess Methods," in *Encyclopedia of Artificial Intelligence*, 1st edition, S. Shapiro (ed.), John Wiley, 1987, pp. 159-171.

M. Newborn, *Computer Chess*, Academic Press, 1975.

J. Schaeffer, "Experiments in Search and Knowledge," Technical Report TR 86-12, University of Alberta, July 1986.

D. J. Slate and L. R. Atkin, "CHESS 4.5−The Northwestern University Chess Program," in *Chess Skill in Man and Machine*, P.W. Frey (ed.), Springer-Verlag, New York, 1983, pp. 92-118.

K. Tsao, H. Li and S. Hsu, "Design and Implementation of a Chinese Chess Program," in *Heuristic Programming in Artificial Intelligence 2*, D. Levy and D. Beal (eds.), Ellis Horwood, London, 1990, pp. 108-118.

J. Tu, *Encyclopedia of Chinese Chess*, Shanghai Cultural Press, 1985.

J. Uiterwijk, Personal communications with *Touch's* author, 3rd Computer Olympiad, Maastricht, The Netherlands, 1991.

C. Ye, "Selective Extensions for Chinese Chess," M.Sc. Thesis, Computing Science Dept., University of Alberta, in preparation, 1992.

A.L. Zobrist, "A New Hashing Method with Applications for Game Playing," Tech. Rep. 88, Computer Science Department, Univ. of Wisconsin, Madison, April, 1970; Also, *Int. Comp. Chess Assoc. J.*, 13(2), 1990, pp. 169-173.