Quiescence Search for Stratego

Maarten P.D. Schadd Mark H.M. Winands

Department of Knowledge Engineering, Maastricht University, The Netherlands

Abstract

This article analyses quiescence search in an imperfect-information game, Stratego. We point out that there are two key problems. The first problem is the search overhead, and the second problem is the limited information gain. A possible solution to the first and second problem is an Evaluation-Based Quiescence Search (EBQS). Experiments show that this solution improves the playing strength of a Stratego program.

1 Introduction

Chess programs perform an exhaustive search on a limited part of the game tree. At a leaf node, a second search is sometimes performed, the so-called quiescence search (QS) [9, 13, 14, 20]. The purpose of this search is that only 'quiet' positions are evaluated (i.e., with no wild swings in the future). Performing such a QS will largely reduce the horizon effect. The need for evaluating quiet positions has been identified in the literature for quite some time [21, 25].

In spite of the fact that quite some research has been done for QS in perfect-information games, the effect of QS in imperfect-information games remains unknown. The imperfect-information game we have chosen is Stratego.[1] Just like in Chess, gaining material is an important factor. A horizon effect may arise for positions where an exchange of material is possible due to capturing. Stratego is therefore a good candidate for applying QS. In this article we will reveal two problems of using QS in Stratego. The first problem is the search overhead, and the second problem is the limited information gain. We propose Evaluation-Based Quiescence Search (EBQS) to solve these problems.

This article will first explain the rules of Stratego in Section 2. Next, the standard search technique for non-deterministic board games, called EXPECTIMAX, is described in Section 3. This technique can be applied to imperfect-information games as well. Thereafter, Section 4 presents QS. We discuss the problems and the possible solution EBQS in Section 5. Finally, Section 6 shows the experiments and Section 7 presents the conclusions.

2 Stratego

Stratego is an imperfect-information zero-sum game. It was developed at least as early as 1942 by Mogendorff. The game was sold by the Dutch publisher *Smeets and Schippers* between 1946 and 1951 [8]. In this section, we will first briefly describe the rules of the game (2.1) and thereafter present related work (2.2). Finally, in Subsection 2.3 the evaluation function is briefly discussed.

2.1 Rules

The following rules are an edited version of the Stratego rules published by the Milton Bradley Company in 1986 [5]. Stratego is played on a 10×10 board. The players, Red and Blue, place each of their 40 pieces in a 4×10 area, in such a way that the back of the piece faces the opponent. The movable pieces are divided in ranks (from the lowest to the highest): Spy, Scout, Miner, Sergeant, Lieutenant, Captain, Major, Colonel, General and Marshal. Each player has also two types of unmovable pieces, the Flag and the Bomb.

Players move alternately, starting with Red. Passing is not allowed. Pieces are moved to orthogonally-adjacent vacant squares. The Scout is an exception to this rule, and may be moved like a Rook in chess. The *Two-Squares Rule* and the *More-Squares Rule* prohibit moves which result in repetition.[2] The lakes in the centre of the board contain no squares; therefore a piece can neither move into nor cross the lakes. Only one piece may occupy a square.

A piece, other than a Bomb or a Flag, may attempt to capture an orthogonally adjacent opponent's piece; a Scout may attempt to capture from any distance. When attempting a capture, the ranks are revealed and the weaker piece is removed from the board. The stronger piece will be positioned on the square of the defending piece. If both pieces are of equal rank, both are removed. The following special rules apply to capturing. The Spy defeats the Marshal if it attacks the Marshal. Each piece, except the Miner, will be captured when attempting to capture the bomb.

The game ends when the Flag of one of the players is captured. The player whose Flag is captured loses the game. A player also loses the game if there is no possibility to move. The game is drawn if both players cannot move.

2.2 Previous Work

Stratego has received limited scientific attention in the past. De Boer [6, 7] describes the development of an evaluation function using an extensive amount of domain knowledge for a 1-ply search. Treijtel [24] created a player based on multi-agent negotiations. Stengård [23] investigates different search techniques for this game. Schadd *et al.* [18] developed a forward-pruning technique for chance nodes ,which was tested in Stratego. At this moment, computers play Stratego at an amateur level [17]. An annual Stratego Computer Tournament[3] is held on Metaforge.[4]

2.3 Evaluation Function

A well-performing evaluation function is an important factor for the playing strength. For our program, we have chosen a material-based approach, partially based on De Boer [6]. The evaluation function is bound to [-1,000, 1,000] which corresponds to losing or winning the game. The piece value is shown in Table 1.

Table 1: Piece values

Spy	Scout	Miner	Sergeant	Lieutenant	Captain
100(10)	10	100	20	50	100
Major	Colonel	General	Marshal	Bomb	Flag
140	175	300	400	75	1,000

When the opponent's Marshal is captured, the value of the Spy is reduced to 10. Furthermore, each position on the board is assigned a value of importance (i.e., increase the distance of the opponent to the own Flag, while decreasing the own distance to the opponent's Flag). The value of a position can be up to 50 points. A bonus is given if information of a piece is hidden. This bonus is set to 30% of the value of the piece. A small random factor is introduced to model the mobility. Furthermore, a variable counts the number of subsequent moves without capturing. For each two non-capture moves, the evaluation score is reduced by 1 point. Using this factor, more risk is taken when nothing happens in the game.

3 EXPECTIMAX

EXPECTIMAX [15] is a brute-force, depth-first game-tree search algorithm that generalizes the MINIMAX concept [27] to non-deterministic games,[5] by adding *chance nodes* to the game tree (in addition to MIN and MAX nodes). At chance nodes, the heuristic value of the node (or EXPECTIMAX value) is equal to the

[2] For details of these rules we refer to the International Stratego Federation (ISF), www.isfstratego.com.

[3] http://www.strategousa.org/wiki/index.php/Main_Page

[4] http://www.metaforge.net/

[5] An imperfect-information game such as Stratego can be treated as a non-deterministic game.

weighted sum of the heuristic values of its successors. For a state s, its EXPECTIMAX value is calculated with the function:

$$\text{EXPECTIMAX}(s) = \sum_i P(c_i) \times V(c_i)$$

where c_i represents the ith child of s, $P(c)$ is the probability that state c will be reached, and $V(c)$ is the value of state c.

We explain EXPECTIMAX in the following example. Figure 1 depicts an EXPECTIMAX tree. In the figure, squares represent chance nodes, regular triangles MAX nodes and inversed triangles MIN nodes. Node A corresponds to a chance node with two possible events, after which it is the MIN player's turn. The value of node A is calculated by weighting the outcomes of both chance events. In this example, $\text{EXPECTIMAX(A)} = 0.9 \times -250 + 0.1 \times 250 = -200$.

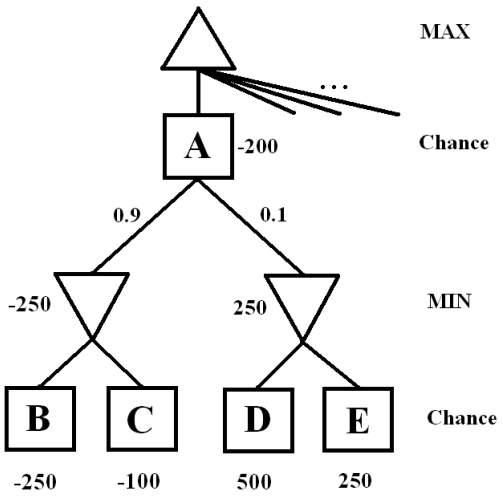

Figure 1: An example EXPECTIMAX tree

The basic idea of EXPECTIMAX is sound but slow [2]. STAR1 and STAR2 exploit a bounded heuristic evaluation function to generalize the $\alpha\beta$ pruning technique to chance nodes [2, 12]. $\alpha\beta$ pruning imposes a *search window* (α,β) at each MIN or MAX node in the game tree. Remaining successor nodes can be skipped as soon as the current node's value is proven to fall outside the search window. STAR1 and STAR2 apply this idea to chance nodes. These methods exploit a bounded evaluation function to calculate theoretical lower and upper bounds for a chance node. After a child is searched, the bounds are updated and it is checked whether the bounds fall outside the window. The difference is that the search cannot return as soon as one successor falls outside the search window. The theoretical lower or upper bound has to fall outside the window to stop the search prematurely at a node.

4 Quiescence Search

Chess programs perform an exhaustive search on a limited part of the game tree. At a leaf node, the so-called quiescence search (QS) can be performed [9, 13, 14, 20]. The purpose of this search is that only 'quiet' positions are evaluated, where no tactical move exists. Performing a QS will largely reduce the horizon effect. The need for evaluating quiet positions has been identified early in the literature [21, 25]. Only evaluating at the desired search depth can under- or overestimate positions. For example in Stratego, a position may look advantageous because the player has a Marshal advantage. However, if the opponent has the possibly to capture this Marshal with a Spy, it renders the position worthless.

Schrüfer [20] formulated the following pseudo code: Let n be a MAX node. The set of n's children is denoted by SUCC(n) = $\{n_1, n_2, ..., n_b\}$, and let TSUCC(n) $\subset$ SUCC(n) denote the set of children reached by *tactical* moves (e.g., capture moves).

In an exhaustive search, the MINIMAX value mm(n) of n is defined by the MINIMAX principle as:

$$\text{mm}(n) = \begin{cases} \text{quiescence value of } n & \text{if } n \text{ is a leaf of the exhaustive search} \\ \max\{\text{mm}(n') \mid n' \in \text{SUCC}(n)\} & \text{if } n \text{ is a non-leaf MAX node} \\ \min\{\text{mm}(n') \mid n' \in \text{SUCC}(n)\} & \text{if } n \text{ is a non-leaf MIN node} \end{cases} \quad (1)$$

In the standard QS only a subset of the children of n, namely TSUCC(n), is considered more closely, and the MINIMAX principle of (1) above then is replaced by the restricted-quiescence principle:

$$\text{qv}(n) = \begin{cases} \max\{\text{eval}(n), \max\{\text{qv}(n') \mid n' \in \text{TSUCC}(n)\} & \text{if } n \text{ is a MAX node} \\ \min\{\text{eval}(n), \min\{\text{qv}(n') \mid n' \in \text{TSUCC}(n)\} & \text{if } n \text{ is a MIN node} \end{cases} \quad (2)$$

where qv(n) denotes the quiescence value of n, and eval(n) denotes the value returned by the evaluation function of the program. This means that if the tactical moves are not profitable, each player has the opportunity to stop the QS at each point of time. Therefore, the QS value at depth 1 is an upper bound for the real QS value.

5 Quiescence Search and Stratego

In this section, we describe the application of QS in Stratego. Section 5.1 discusses problem 1, the search overhead in the QS. Section 5.2 shows an analysis of problem 2, the effectiveness of QS in Stratego. Section 5.3 describes Evaluation-Based Quiescence Search (EBQS).

5.1 Problem 1: Search Overhead of QS

The first problem we discuss is the possible search overhead of QS for Stratego. The game has the property of not turning quiet for many moves in certain positions. Imagine a position where Red has a General inside the opponent's base area, which almost completely filled with unknown pieces. If at this position a QS is applied, the search will not return for a long time. The reason for this is the presence of chance nodes, which are involved each time a capturing takes place. Each time that the red General attacks a blue piece, or is getting attacked by an unknown blue piece, the majority of events result in the fact that the General is still on the board. The search will have to continue until either (1) no new piece can be reached, or (2) all unknown pieces are able to capture the General. At a position with many blue unknown pieces next to each other, a large amount of time has to be invested. Typically, the more plies are searched, the higher the evaluation will be. Due to the fact, that the opponent can choose not to re-capture, deep searches will likely not influence the QS value. Furthermore, a high QS value might not propagate to the leaf node because it is averaged in a chance node with other outcomes. Thus, a large effort is taken with possibly little result.

As a possible solution, we implemented a depth limit for QS. A choice is to limit the number of chance nodes allowed in the search. Capturing of known pieces is faster and takes only little overhead. When the search reaches the limit of chance nodes, the position is evaluated and the score is returned. We also forward prune the Scout moves, which contribute significantly to the branching factor without making much gain. Table 2 shows the overhead of the QS limited to one chance node. A search up to 4 ply was executed on 300 begin, middle and endgame positions, created during selfplay.[6]

Table 2: Quiescence Search Overhead

Ply	Normal	QS Limit 0	QS Limit 1	QS Limit 2
1	8,413	13,302	60,944	434,385
2	231,591	369,279	955,552	4,602,505
3	2,187,222	3,434,674	15,506,365	101,132,268
4	48,610,457	79,610,578	156,966,429	684,607,920

We observe that the majority of nodes are allocated in the QS (e.g., 97.8% with depth limit 2 for a 3-ply search). If the QS would not have been limited, a substantially larger number of nodes would be investigated.

[6] All test positions for Stratego can be downloaded at http://www.personeel.unimaas.nl/Maarten-Schadd/TestSets.html

5.2 Problem 2: Effectiveness of QS

In classic games, such as chess or checkers, capturing a piece may be quite advantageous (i.e., material is the dominant factor). Due to this, QS has a large impact in the playing strength. In Stratego, capturing might lead to the loss of an own piece. On top of this, evaluating a capturing move is twelve times as expensive, due to the fact of considering each rank that the unknown opponent's piece could be.

One might think of reducing the search effort by making a generalization. In each chance node only 3 cases might be considered (losing, draw, winning). However, revealing the position of the Marshal can render losing a piece quite valuable, while revealing lower ranked pieces are not advantageous. This means that an expensive search may result in a small information gain.

In the next experiment, we tested the regular QS for a chance node limit of 1. Table 3 shows the difference range of QS value compared to the normal evaluation value. *Bin* is the range of change of the QS value. For example, Bin 0 indicates that the QS returned a value change between -10 (excluding) and 0 (including).

Table 3: QS Value Difference for Chance-Node Limit 1

Bin	-200	-190	-180	-170	-160	-150	-140	-130	-120	-110
Amount	269	50	19	34	7	2	4	24	41	42
Bin	-100	-90	-80	-70	-60	-50	-40	-30	-20	-10
Amount	71	49	37	79	75	90	202	489	1,010	1,439
Bin	0	10	20	30	40	50	60	70	80	90
Amount	29,590	19,836	2,510	518	1,278	567	102	255	463	416
Bin	100	110	120	130	140	150	160	170	180	190
Amount	169	113	34	5	1	4	5	3	10	9

We see in Table 3 that in the majority of situations a value close to 0 is returned. This means that the gain of QS is small. Moreover, in more than half of the cases, the QS value is not even used, because it is lower than the evaluation of the position under consideration.

Now the question remains, what the information gain is of the QS. For this we counted the number of times the QS value was used, which is given in Table 3. In only 43.9% of the cases the QS value proved to be useful. If the QS value is useful, the evaluation value is changed by 13.3 points on average ($\approx$ value of a Scout).

5.3 Solution: Evaluation-Based Quiescence Search

To overcome the problems described in the previous sections, a new approach for QS has to be used. Instead of investing major efforts in exploring non-quiet positions in the tree, Evaluation-Based Quiescence Search (EBQS) can be applied. This method is inspired by static exchange evaluators [16]. This method should be fast to compute and give an accurate estimation of the real QS value.

For calculating the EBQS value we will evaluate each local encounter between pieces separately. We distinguish between different levels of detail as follows. Level 0: the result of encounters between two known pieces is known. Level 1: there is one unknown piece involved. Level 2: two own pieces against 1 unknown piece of the opponent, and Level 3: two own pieces against 2 unknown pieces of the opponent. Higher levels are possible, but rarely occur during a Stratego game.

In this article we will focus on the Level 0 and 1 approach. The pseudo code of EBQS is given in Algorithm 1. This algorithm calculates the quiescence value for each of the pieces separately, using the *qsEval()* function. This function is straightforward to implement in the case that both pieces are known (Level 0). With one unknown piece (Level 1), *qsEval()* has to calculate the probability based on the remaining unknown pieces. To increase the speed of the calculations, we note that the *qsEval()* function can use a hash table.

In general, this function computes a quiescence value for the complete board, without taking re-capturing into account. Using this fast function instead of an actual QS allows the search to overcome the horizon effect without performing an actual QS.

Algorithm 1 Evaluation-Based Quiescence Search

```
int EBQS(position)

int qsValue=0;

for all Pieces p in position do
  int min = max = 0;
  for all Neighbours(p) n do
    if IsOpponent(n) then
      int v = qsEval(p, n);
      if v > max then
        max = v;
      end if
      if v < min then
        min = v;
      end if
    end if
  end for

  if RedPiece(p) then
    qsValue += max;
  end if
  if BluePiece(p) then
    qsValue -= min;
  end if
end for

return qsValue
```

6 Experiments

We implemented an EXPECTIMAX engine for Stratego, enhanced with the STAR1 and STAR2 pruning algorithms [2, 10]. Furthermore, the History Heuristic [19], Killer Moves [1], Transposition Tables [9, 22] and StarETC [26] are incorporated. Null-Move [3], ProbCut [4] and ChanceProbCut [18] are used as well.

Before we test EBQS, we first measured the performance of the classic QS, using selfplay experiments. The time setting was one second per move. The results are given in Table 4, for different chance node limits.

Table 4: Games won by QS versus No QS, 1,000 games played

	QS Limit 0	QS Limit 1	QS Limit 2	QS Limit 3	QS Limit 10	QS No limit
No QS	478	501	503	505	515	513
QS	522	499	497	495	485	487

We can make two observations from this table. (1) QS does not improve the playing strength when chance nodes are included. The regular search was better or equal to each setting, except for limit 0. (2) Without depth limitation, only a slight decrease in performance is measured. The reason for this is that the number of positions where the search explodes is rather low. This means that a suboptimal move is played only in a few positions during the complete game. Considering the large number of moves during a Stratego game, a player has enough time to correct his mistake. Even without a depth limitation on the number of allowed chance nodes during QS, only a small decrease in playing strength is measured.

In the next experiment the performance of EBQS was tested. The time setting was one second per move. The programs used initial setups in the same way as described in Subsection 5.2. The results are given in Table 5. We observe an improvement, which is significant. A win rate of 52.4 % is achieved for 5,000 games against a normal search, and 56.3% is achieved against the QS search with limit 0. This means that EBQS is able to compute tactical information without a large time overhead. The results are quite good, because

large improvements cannot be expected in the EXPECTIMAX framework [11, 12].

Table 5: Performance of EBQS

No QS	2,382	QS Limit 0	2,184
EBQS	2,618	EBQS	2,816
Win rate	52.4%	Win rate	56.3%

7 Conclusions

In this article we pointed out two problems of quiescence search (QS) that occur in Stratego. The first problem is the overhead of QS. The second problem is that information gained during the QS search is only for a limited amount of cases useful.

To overcome both problems, we designed an algorithm to compute an Evaluation-Based Quiescence Search (EBQS) value. The value estimates the effect of tactical moves in a leaf node. We showed that EBQS is able to improve the playing strength of our Stratego program, while normal QS is not able to do so.

This research showed that EBQS may be used as a replacement for normal QS. However, this EBQS can still be improved. At this point of time, re-capturing is not considered (Level 2). Including this, and possibly even longer capturing sequences, a larger improvement might be obtained. We may conclude that EBQS is an alternative for QS in Stratego.

Acknowledgements

This work is funded by the Dutch Organisation for Scientific Research (NWO) in the framework of the project TACTICS, grant number 612.000.525.

References

[1] S. G. Akl and M. M. Newborn. The Principal Continuation and the Killer Heuristic. In *1977 ACM Annual Conference Proceedings*, pages 466–473, ACM Press, New York, NY, USA, 1977.

[2] B. W. Ballard. The *-Minimax Search Procedure for Trees Containing Chance Nodes. *Artificial Intelligence*, 21(3):327–350, 1983.

[3] D. F. Beal. Experiments with the Null Move. In D. F. Beal, editor, *Advances in Computer Chess 5*, pages 65–89. Elsevier Science Publishers B.V., 1989.

[4] M. Buro. ProbCut: An Effective Selective Extension of the Alpha-Beta Algorithm. *ICCA Journal*, 18(2):71–76, 1995.

[5] Milton Bradley Co. *Stratego Instructions*. Milton Bradley Co., 1986. Obtained at http://safemanuals.com/.

[6] V. de Boer. Invincible. A Stratego Bot. Master's thesis, Delft University of Technology, The Netherlands, 2007.

[7] V. de Boer, L. J. M. Rothkranz, and P. Wiggers. Invincible - A Stratego Bot. *International Journal of Intelligent Games & Simulation*, 5(1):22–28, 2008.

[8] District Court of Oregon. *Estate of Gunter Sigmund Elkan, vs. Hasbro, INC. et al.*, 2005. Case No. 04-1344-KI.

[9] R. D. Greenblatt, D. E. Eastlake, and S. D. Crocker. The Greenblatt Chess Program. In *Proceedings of the AFIPS Fall Joint Computer Conference 31*, pages 801–810, 1967. Reprinted (1988) in *Computer Chess Compendium* (ed. D. N. L. Levy), pp. 56-66. B. T. Batsford Ltd., London, UK.

[10] T. Hauk. Search in Trees with Chance Nodes. Master's thesis, University of Alberta, Edmonton, Canada, 2004.

[11] T. Hauk, M. Buro, and J. Schaeffer. *-Minimax Performance in Backgammon. In H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu, editors, *Computers and Games, CG 2004*, volume 3846 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag, 2006.

[12] T. Hauk, M. Buro, and J. Schaeffer. Rediscovering *-minimax search. In H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu, editors, *Computers and Games, CG 2004*, volume 3846 of *Lecture Notes in Computer Science*, pages 35–50. Springer-Verlag, 2006.

[13] H. Kaindl. Dynamic Control of the Quiescence Search in Computer Chess. In R. Trappl, editor, *Cybernetics and Systems Research*, pages 973–978, Amsterdam, The Netherlands, 1982.

[14] T. A. Marsland. A Review of Game Tree Pruning. *ICCA Journal*, 6(3):16–19, 1986.

[15] D. Michie. Game-Playing and Game-Learning Automata. In L. Fox, editor, *Advances in Programming and Non-Numerical Computation*, pages 183–200, Pergamon, New York, USA, 1966.

[16] F. M. H. Reul. *New Architectures in Computer Chess*. PhD thesis, Universiteit van Tilburg, Tilburg, The Netherlands, 2009.

[17] I. Satz. The 1st Computer Stratego World Championship. *ICGA Journal*, 31(1):50–51, 2008.

[18] M. P. D. Schadd, M. H. M. Winands, and J. W. H. M. Uiterwijk. CHANCEPROBCUT: Forward Pruning in Chance Nodes. In P. L. Lanzi, editor, *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, pages 178–185, 2009.

[19] J. Schaeffer. The History Heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[20] G. Schrüfer. A Strategic Quiescence Search. *ICCA Journal*, 12(1):3–9, 1989.

[21] C. E. Shannon. Programming a Computer for Playing Chess. *Philosophical Magazine*, 41(7):256–257, 1950.

[22] J. D. Slate and L. R. Atkin. CHESS 4.5: The Northwestern University Chess program. In P. W. Frey, editor, *Chess Skill in Man and Machine*, pages 82–118, New York, USA, 1977. Springer-Verlag. Second Edition.

[23] K. Stengård. Utveckling av Minimax-Baserad Agent för Strategispelet Stratego. Master's thesis, Lund University, Sweden, 2006. In Swedish.

[24] C. Treijtel and L. J. M. Rothkrantz. Stratego Expert System Shell. In Q. H. Mehdi and N. E. Gough, editors, *GAME-ON 2001*, pages 17–21, 2001.

[25] A. M. Turing. Digital Computers Applied to Games. *Faster than Thought*, pages 286–297, 1953.

[26] J. Veness and A. Blair. Effective Use of Transposition Tables in Stochastic Game Tree Search. In A. Blair, S-B. Cho, and S. M. Lucas, editors, *IEEE Symposium on Computational Intelligence and Games (CIG 2007)*, pages 112–116, 2007.

[27] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen 100*, pages 295–320, 1928. Translated by Bargmann (1959).