Solving 5 × 5 Amazons

Martin Müller
Department of Computing Science
University of Alberta
Edmonton, Canada
mmueller@cs.ualberta.ca

Abstract

We show that the game of Amazons played on a 5 × 5 board with four playing pieces each is a first player win. The proof uses a standard minimax game tree search with several enhancements that reduce the depth and the width of the required search tree. We develop divide-and-conquer techniques which are inspired by combinatorial game theory, but use only integer-valued bounds on the value of subgames. After partitioning an Amazons board into independent subgames, upper and lower bounds on the game value are computed for areas which contain amazons of both players.

Key words: Amazons, game tree search, perfect play

1 The Game of Amazons

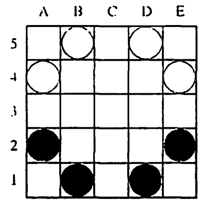

Figure 1: Starting Position in 5 × 5 Amazons

The game of Amazons is played on a square board by two players, Black and White. Each player controls four playing pieces, called *amazons*, which move like a chess queen. After each move an amazon shoots an arrow, which travels in the same way as a chess queen moves. The point where an arrow lands is *burned off* the playing board, reducing the effective playing area. Neither amazons nor arrows can travel across a burned off square or over another amazon. The last player able to make a move wins the game. In this paper, we consider play on a 5 × 5 board, with the starting position shown in Figure 1. We assume that Black moves first.

1.1 Partitioning the Board

The aim of partitioning a board is to separate a position into smaller pieces which can be analyzed independently, or nearly independently from each other. A basic board partition is given by the 8-connected components of all points on the board which have not been burned off by arrows. In Figure 2, partitioning the position shown on the left results in the areas a, b and c in the picture in the middle. An improved board partitioning can be achieved with the help of *blockers*, amazons that further divide a component into two or more regions such that some of these regions are only adjacent to amazons of one color [2]. The picture on the right in Figure 2 shows the resulting five areas $a \ldots e$.

Regions on the board can be classified into four categories, depending on the amazons of that region.

1. A region that contains at least one empty square, and contains or is blocked by amazons of both colors is called an *active area* or *active region.*

2. A region that contains at least one empty square, and contains or is blocked by amazons of a single color is a *territory* for that player.

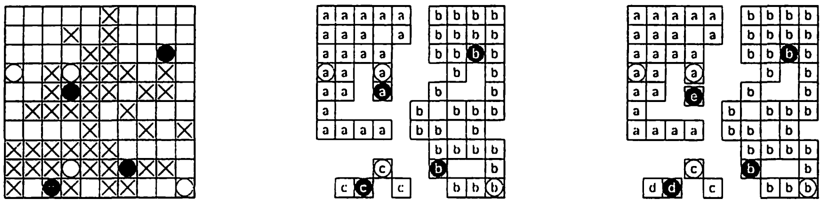

Figure 2: Board position and basic decomposition, from [2]

3. A *neutral region* contains no amazons.
4. A *dead region* contains only amazons, no empty squares.

The neutral and dead regions are irrelevant since there are no possible moves there. In territories, the main question is how many moves the controlling player can make there. In active regions, both players compete for the right to make as many moves as possible.

2 An Evaluation Function for an Amazons Solver

Arrow-S is a specialized program for solving Amazons positions. The program resolves only wins and losses. it does not try to determine by how many extra moves a player can win. Using this solver 5×5 Amazons was proven to be a first player win. The search space of this game is about 9.92×10^{12} positions. As the experiments in Section 4 indicate, a brute force search of this game takes a long time. Therefore this paper describes rules to recognize wins and losses earlier in the game, as well as an exact method to prune some types of moves.

2.1 Estimating the Game Value

In the following, we will assume that positive integers represent a number of moves for Black, and negative integers represent moves for White. The program computes an upper and a lower bound on the difference (black moves - white moves) for each area on the board. Sometimes, for example in simple territories, these bounds coincide and therefore give the exact value of a region. Bounds on the overall game value are computed by summing up the bounds of each region. If the overall lower bound is greater than zero, the game is a sure win for Black, and if the upper bound is below zero it is a win for White. Likewise, a lower bound of exactly zero is a win for Black if it is White's turn to play, and an upper bound of zero means a win for White if it is Black's turn to play.

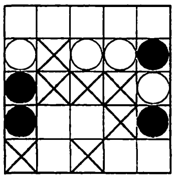

Figure 3: White to play, Black wins

Example: White is to play in Figure 3. In this position, the program can prove a win for Black by its static evaluation. The black territory at the bottom is worth exactly 5 moves, while White can make at most 5 moves at the top. White has to move first, so Black is guaranteed to get the last move and therefore wins.

2.2 Counting Moves in Territories

The number of empty squares is an obvious upper bound on the number of moves in a territory, and often it coincides with the precise value. However, *defective* territories exist, which cannot be completely filled in [2]. There are several ways of computing the size of a territory or at least finding a lower bound on its size, including exhaustive search and using a database of defective areas [2]. Within our solver, we currently use the simplest and fastest method, a *plodding* walk without any backtracking, to compute a lower bound on the number of moves a player can make in a territory. A *plod* [1] is a move sequence where an amazon moves to an adjacent square and fires back at its origin square. Such a plod is very quick to implement by just marking the squares visited so far. In the case when there are two or more nonmarked adjacent empty squares, a simple heuristic determines which of these squares an amazon should visit next. This method is good enough to completely fill most of the small nondefective territories that occur on small board sizes, such as 5×5 or 6×6. In this case, the lower bound coincides with the upper bound and the exact value of an area has been determined.

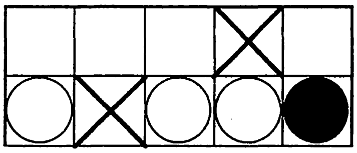
Figure 4: Sure move for Black going first

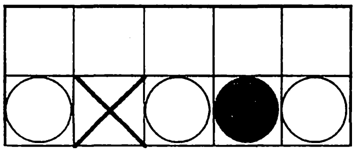
Figure 5: Sure moves for Black going second, 3 AES

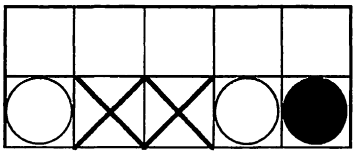
Figure 6: Sure move for Black going second, 2 AES

2.3 Counting Moves in Regions Involving both Players

Consider an active region which involves amazons of both players. When playing out such a region, the only result that matters for the overall outcome is the difference in the number of black moves minus the number of white moves made in this region. The goal is to identify easy to compute upper and lower bounds on that value and use such bounds to estimate the overall outcome. The basic lower bound for the value of a region with n empty squares is $-n$, for the case where White can make all the moves there, and the upper bound is n in

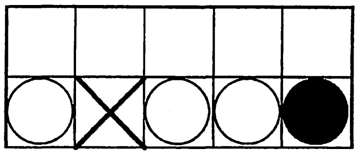
Figure 7: No safe move for Black going second

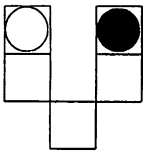
Figure 8: Safe moves far away from opponent

case Black gets all the moves. These bounds can be tightened in two ways:

1. By finding safe moves for a player that the opponent cannot eliminate even if the opponent moves first.
2. By finding a move for the player who goes first.

In each case a bound is improved by 2, since the difference in the number of moves changes from $n : 0 = n$ to $(n-1) : 1 = n-2$. However, the second rule can only be used in the case where the player moves first in a region.

2.3.1 Finding safe moves for an amazon going first.

Each amazon that has an adjacent empty square (AES) can make at least one move by moving there and shooting back. For example, the black amazon in Figure 4 has one AES.

2.3.2 Finding safe moves for an amazon going second.

With each move, the opponent can take away at most two AES of an amazon, one by moving an amazon there and another by shooting. So an amazon with three or more AES can always make a move, even if the opponent goes first. In Figure 5, the black amazon has three AES. An amazon with only one or two AES might still have a safe move in

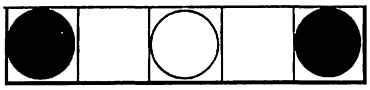
Figure 9: One amazon can block two opponents

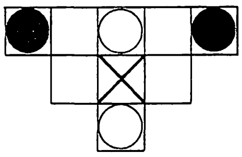
Figure 10: Dependency between safe moves

the cases where the opponent either cannot reach all AES or, in the process of reaching them, has to move one of its amazons away from an adjacent square, thereby creating a new AES. For example, in Figure 6 the black amazon has only two AES but the only opponent amazon that can block both squares is adjacent to the black amazon and cannot control all three squares including its origin square. However, in Figure 7 the white amazon in the middle of the board can completely block Black. In Figure 8 each amazon has a safe move because the opponent is too far away to block it.

2.3.3 Combining safe moves for several amazons

If there are several amazons in the same region, their moves can easily interact, as the following two cases illustrate.

Dependencies between several amazons The white amazon in Figure 9 has a safe move: even though each of its AES is easily blockable by the opponent, they cannot both be blocked in one move. However, this argument cannot be combined for several amazons, as shown in Figure 10. Each white amazon taken by itself has a safe move, but they are dependent, and if Black moves first then White cannot make two safe moves in this area.

Move going first If two amazons are in the same area, one opponent might be able to block both, as shown in Figure 9. If there are several amazons that can make a move going first, only one in three amazons can be sure to make a move, e.g. the first and forth amazon of the same color, but not the second and third one. When amazons are not in the same area, every other amazon can move.

We could not find a good way to statically check these dependencies, so in our current implementation only one amazon per area of each player is used to improve the bound.

2.4 Territories and Active Areas with a Common Blocker

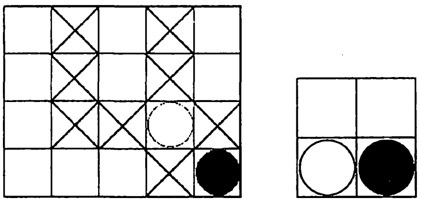
Figure 11: Territories with a common blocker, and active region with value 0

In the case where a blocker blocks off two or more territories at once, the blocker in general cannot fill in more than one territory. For example in Figure 11 on the left, the white amazon blocks off three separate territories but will be able to fill only one of them. Likewise, the blocker cannot move into an adjacent active area without giving up its moves in all the territories. We use the following simple greedy heuristic: First, all non-blocking amazons will plod to make as many moves as possible. Then blockers are allowed to plod in the remaining areas of each territory or are used in an adjacent active area, and for each blocker its single most valuable action is chosen for improving the bound of that region.

3 The Search Process

A 5 × 5 Amazons board with 4 amazons each initially has 17 empty squares, so the maximum game length is 17 ply. The branching factor is 262 for the first move, and typically about 80 after 5 moves. Initial experiments indicated that the search space was too large to complete a brute-force search in a reasonable time. Therefore, a specialized Amazons solver *Arrow-S* was built along the lines described above by extending and modifying the heuristic Amazons-playing program *Arrow* [2]. The rules to compute bounds were added as a first step to the static evaluation function. If a position can be statically proven to be a win or a loss, it obtains a value of $\pm\infty$ (in practice, a large integer is used). For other positions, the normal heuristic evaluation function of *Arrow* is used to achieve a good move ordering in conjunction with an iterative deepening search. The search continues at increasing depths until a win or a loss is found.

Arrow-S implements a pruning rule that removes provably irrelevant moves from the search. The heuristic program *Arrow* uses many more pruning rules, including heuristics that we think are almost always sound. However, all these heuristic rules are disabled in *Arrow-S* to ensure correctness. In this paper we describe only the exact rule used in *Arrow-S*.

In regions where upper and lower bounds coincide, the local score is exactly known and it is an integer. Most of these regions are territories, where only one player can move. But some other regions can be recognized as having a constant value, such as the position of value 0 shown in Figure 11 on the right. For such regions, move generation can be greatly restricted. It is sufficient to generate a single move for each player globally for such regions. In terms of combinatorial game theory, moves in a territory have temperature -1. There are only two reasons to generate one such move: These may be the only moves left for a player, or other moves exist but are even worse because of a *zugzwang* situation [2].

4 Result: Solving 5×5 Amazons

We have established that 5×5 Amazons is a first-player win, and have found four different winning moves. Among the winning moves, the two moves 1.B1-B3xD3 and 1.B1-D3xB3, as well as the symmetric 1.D1-B3xD3 and 1.D1-D3xB3, were found to be wins with a search depth of only 6 ply. For both of these moves, a simple blocking strategy proved to be sufficient to win the game. Figure 12 shows the principal variation after 1. B1-B3xD3.

For the other easy opening, our principal variation is 1. B1-D3xB3. 2. A4-A3xC1 3. D1-D2xB2 4. B5-C4xB4 5. D2-C3xE5 6. E4-E3xE4 7. E2-D2xE2. The play is similar to the 1. B1-B3xD3 opening. Black ends up with a five point area at the bottom, which is sufficient to win against the five empty points left at the top.

Besides these two "easy" wins and their symmetric cases, two more winning moves were found at search depths of 8 and 10 ply, 1. B1-B4xB2 and 1. D1-D4xB2. These are the starting moves in the games *f1* and *f2* analyzed below. We are currently working with Thomas Lincke to analyze other opening moves and create a complete opening book for 5×5 Amazons.

Compared to our previous program *Arrow*, the specialized solver *Arrow-S* recognizes wins and losses several ply earlier, which leads to a huge reduction in the tree size. For example, variations following a bad move are now disproven very quickly by creating a large territory for the opponent, and using bounds to prove a loss.

4.1 The Effect of Improved Bounds on Search Efficiency

As a test set, positions were taken from three real games, two 5×5 games, *f1* and *f2*, as well as one 6×6 game, *f3*. These games were email games played at the rate of one move per day by our heuristic program *Arrow* against an expert amazons player, Thomas Lincke. Games that ended in resignation were finished by *Arrow* self play.

The tests start with easy positions near the end of the games, and move forward in the game as long as the computation time is within about 10 hours. To assess the impact of our new methods, four versions of the solver are compared:

A Full evaluation: in addition to computing local bounds, use a global combination of possible single local moves to improve the bounds on the overall position.

B Use bounds on territories (see method C below) as well as local bounds on active areas by finding safe moves on adjacent empty squares.

C Bounds on moves in territories only. A player wins if the surplus in territory is greater than

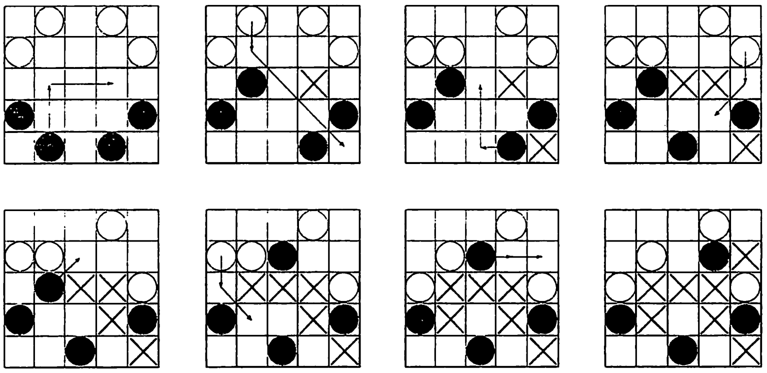

Figure 12: 1.B1-B3xD3 2.B5-B4xE1 3.D1-C1xC3 4.E4-E3xD2 5.B3-C4xB3 6.A4-A3xB2 7.C4-D4xE4

the number of moves on the rest of the board, or equal if it is the opponent's turn to play.

D Brute force search. No bounds are calculated. A player loses if there are no legal moves.

Tables 1 to 3 show the number of nodes, solution time in seconds on a Macintosh G4/400 and search depth (in the last four columns) for each position in the test set and for each type of test. Figures 13 to 15 plot the data for the number of nodes (on a logarithmic scale) and search depth. Graphs for solution time are omitted since they are very similar to those for number of nodes.

The results show big differences between problems, indicating the large variety of positions that come up even on small boards in this game. Some positions, mainly close games with little territory, are hard for any kind of search. Games are much easier for bounds-based methods if they are lop-sided, or if they are easy to evaluate because both players have large territories and/or many safe moves in active areas. In game *f1*, the position after move 2 is much easier to prove for the strong algorithms than the position after move 3.

5 Summary and Outlook

We described some techniques that reduce the depth and the width of the search tree for solving Amazons on small boards. The most important technique is the computation of upper and lower bounds for the value of local areas on the board, which makes it possible to prove overall wins and losses early in the game. Using these techniques we have shown that 5 × 5 Amazons is a first-player win. A related question posed by Elwyn Berlekamp is to try to determine the temperature of the starting position. A first step would be to determine whether the first player can win by an extra move. We suspect that the answer is "no".

An open problem is finding safe moves for several amazons in the same region at once. We could not find a good solution, except for the brute-force approach of doing a specialized local search. While there is clearly room for further refinements of our current techniques, we do not believe that they are strong enough to solve the 6 × 6 case in a reasonable amount of time. A conservative extrapolation of the data for game f3 would put the effort of solving the game at over 100.000 hours of computing time. Therefore, as future work we want to investigate the use of databases of defective territories and especially of combinatorial game values for active areas, in order to further enhance the solver.

References

[1] E. Berlekamp. Sums of $N \times 2$ Amazons. In *Institute of Mathematics Statistics Lecture Notes*, number 35 in Monograph Series, pages 1–34, 2000.

[2] M. Müller and T Tegos. Experiments in computer Amazons. In R. Nowakowski, editor, *More Games of No Chance*. Cambridge University Press, 2001. To appear.

Test pos.	Nodes A	Nodes B	Nodes C	Nodes D	Sec A	Sec B	Sec C	Sec D	A	B	C	D
1: move 15	1	· 3	3	3	0.1	0.1	0.1	0.1	0	1	1	1
2 move 14	1	1	11	12	0.1	0.1	0.1	0.1	0	0	2	2
3: move 13	3	3	27	28	0 1	0.1	0.1	0.1	1	1	3	3
4. move 12	15	15	107	116	0.1	0.1	0.2	0.2	2	2	4	4
5: move 11	42	42	347	461	0.2	0.2	0.2	0.2	3	3	5	5
6: move 10	65	67	767	1,184	0.2	0.2	0.2	0.3	4	4	6	6
7. move 9	1,080	1,156	4,973	6,977	0.3	0.3	0.5	0.6	5	5	7	7
8. move 8	3,011	3,363	9,532	16,492	0.4	0.5	0.8	1.1	6	6	8	8
9: move 7	31,212	34,767	134,932	167,001	2.6	3.1	7.8	9.1	7	7	9	9
10: move 6	254,205	260,247	781,168	1,105,037	19.5	20.4	47.6	61	9	9	11	11
11: move 5	1,484,466	1,828,324	4,082,533	7,194,864	106	133	239	378	10	10	12	12
12: move 4	10,499,200	9,289,780	33,998,550	47,807,133	899	801	2454	3244	11	11	13	13
13: move 3	65,214,371	72,813,193	191,746,197	268,676,881	5596	6230	13350	19034	12	12	14	14
14: move 2	2,121,307	21,514,907	236,930,885	-	340	3043	23303	-	7	9	11	-
15: move 1	427,586,440	-	-	-	50722	-	-	-	10	-	-	-

Table 1: Results for 5 × 5 test game *f1*

Test pos	Nodes A	Nodes B	Nodes C	Nodes D	Sec A	Sec B	Sec C	Sec D	A	B	C	D
1: move 16	1	1	1	3	0.1	0.1	0.1	0.1	0	0	0	1
2: move 15	1	1	1	6	0.1	0.1	0.1	0.1	0	0	0	2
3: move 14	1	1	1	31	0.1	0.1	0.1	0.1	0	0	0	3
4: move 13	1	1	1	61	0.1	0.1	0.1	0.2	0	0	0	4
5. move 12	1	1	1	300	0 1	0 1	0.1	0.2	0	0	0	5
6: move 11	1	1	1	456	0.1	0.1	0.1	0.2	0	0	0	6
7: move 10	1	1	1	2,146	0.1	0.1	0.1	0.3	0	0	0	7
8: move 9	1	1	1	3,053	0.1	0.1	0.1	0.3	0	0	0	8
9 move 8	1	3	3	6,660	0.1	0.1	0.1	0.5	0	1	1	9
10 move 7	308	309	375	22,626	0 2	0 2	0.2	1.3	4	4	4	10
11: move 6	356	355	3,012	350,614	0.2	0.2	0.5	15.5	3	3	5	11
12 move 5	4,929	5,192	19,681	545,337	0.8	0.8	2	25.3	4	4	6	12
13: move 4	19,519	45,137	275,975	24,157,502	3.3	6.2	29.8	1502	5	5	7	13
14 move 3	364,738	436,569	14,664,885	78,549,048	47.3	56.8	1267	5582	6	6	10	14
15. move 2	1,439,614	1,484,517	248,739,140	-	257	262	26070	-	7	7	11	-
16: move 1	54,371,895	55,256,136	-	-	6822	6947	-	-	8	8	-	-

Table 2: Results for 5 × 5 test game *f2*

Test pos.	Nodes A	Nodes B	Nodes C	Nodes D	Sec A	Sec B	Sec C	Sec D	A	B	C	D
1. move 19	1	1	317	858	0.1	0.1	0.1	0.2	0	0	4	6
2: move 18	11	11	1,772	4,881	0.1	0.1	0.3	0.5	1	1	5	7
3: move 17	460	584	5,638	26,784	0.2	0.2	0.5	1.4	4	4	8	10
4: move 16	2,497	5,130	32,480	95,831	0.4	0.7	2.4	5.6	5	5	8	11
5: move 15	16,329	33,847	151,798	618,549	1.6	3.4	10.4	31.3	6	6	10	12
6: move 14	78,647	177,482	936,268	3,638,219	7.9	17.6	64.5	220	7	7	11	13
7: move 13	1,418,839	1,939,837	6,670,508	19,857,550	132	175	520	1379	8	10	12	14
8: move 12	3,167,102	4,563,832	58,760,467	193,826,984	333	468	5105	16303	9	9	13	15
9: move 11	17,310,435	22,782,667	141,756,738	-	1823	2399	12605	-	10	10	14	-

Table 3: Results for 6 × 6 test game *f3*

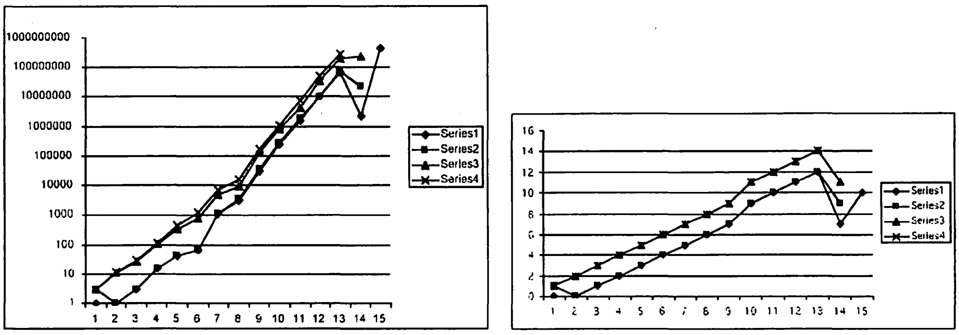

Figure 13: 5×5 game *f1* nodes searched and search depth

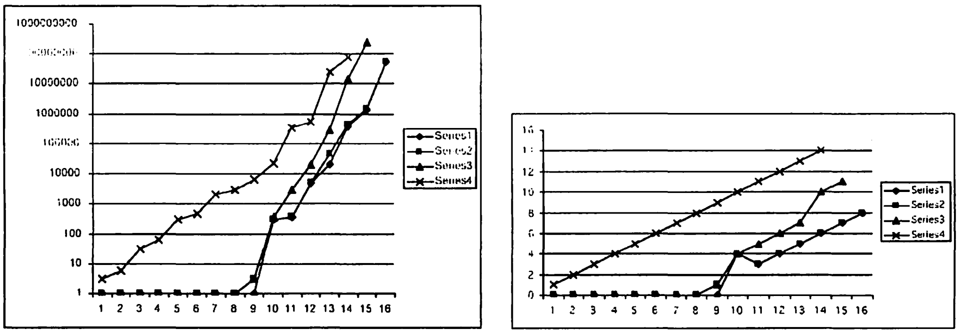

Figure 14: 5×5 game *f2* nodes searched and search depth

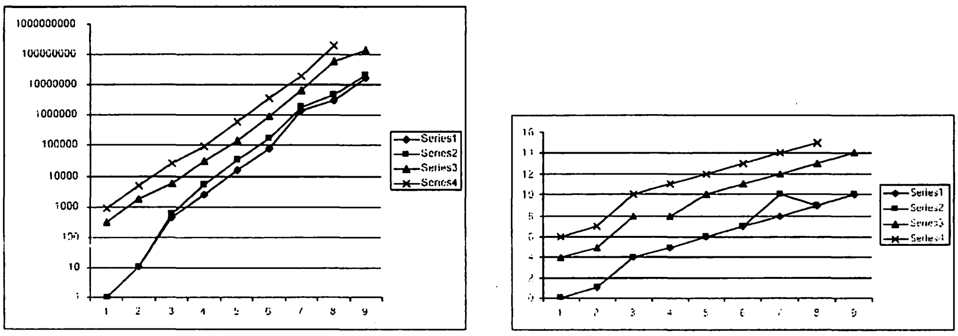

Figure 15: 6×6 game *f3* nodes searched and search depth