An Enhanced Solver for the Game of Amazons

Jiaxing Song and Martin Müller

***Abstract*—The game of *Amazons* is a modern board game with simple rules and nice mathematical properties. It has a high computational complexity. In 2001, the starting position on a 5×5 board was proven to be a first player win. The enhanced *Amazons* solver presented here extends previous work in the following five ways: by building more powerful endgame databases, including a new type of databases for so-called blocker territories, by improving the rules for computing bounds on complex game positions, by local search to find tighter local bounds, by using ideas from combinatorial game theory to find wins earlier, and by using a df-pn based solver. Using the improved solver, the starting positions for *Amazons* on the 4×5, 5×4, 4×6, 5×6, and 4×7 boards were shown to be first player wins, while 6×4 is a second player win. The largest proof, for the 5×6 board, is presented in detail.**

***Index Terms*—- Artificial intelligence, computational and artificial intelligence, combinatorial mathematics, heuristic algorithms, search methods, search problems algorithms.**

I. Introduction

A. Games and Artificial Intelligence

SINCE the beginnings of Artificial Intelligence (AI) in the 1950s, the relationship between games and AI research has been a reciprocal one. Games have finite state spaces, well defined rules and quantifiable goals. They, therefore, offer ideal testbeds for AI research. This leads to competitive opponents for human game players and also often offers interesting insights into the games.

The game of *Amazons* is a modern board game with simple rules yet high complexity [1]. With a typical branching factor in the hundreds of moves, *Amazons* is an ideal testbed for selective search algorithms. Since the board naturally breaks down into independent subgames as a game proceeds, the game also provides a nice test domain for applying combinatorial game theory.

B. Outline

This paper presents several techniques for solving small *Amazons* boards and applies them to determine the game-theoretic result on several small rectangular board sizes. It is organized as follows: Section I motivates AI research in games, introduces the game of *Amazons*, and summarizes related work and the contributions of this paper. Section II describes the methodology used for solving *Amazons*, including essential background on combinatorial game theory, and introduces tightened and relaxed bounds on the value of a game. Section III discusses the three types of local game databases built, with a focus on the new *blocker territory databases*. Section IV gives details on how bounds on the value of each type of area are computed and combined for solving a full-board *Amazons* position. Section V contains experimental results of the solver, including details of the 5×6 solution, results for test cases from harder board sizes, and statistics on the different types of local areas. The final Section VI discusses ideas for further improving the *Amazons* solver.

C. Contributions

The following are the main contributions of this paper:

- A strong evaluation function for computing correct bounds on the value of small full-board *Amazons* positions based on many types of local analysis.
- A df-pn based solver for *Amazons* which utilizes the evaluation components above and improves search efficiency, often by orders of magnitude.
- A technique for computing, storing and using databases of *blocker territories*.
- Two new static rules for improving the bounds on *active* local areas.
- A notation for expressing *tightened* and *relaxed* bounds on the value of a game.
- Techniques for exploiting knowledge about infinitesimals which often occur in *Amazons*, including an application of subzero thermography.
- An extensive empirical evaluation of the new solver.
- Solving the initial positions of *Amazons* on 4×5, 5×4, 4×6, 5×6 and 4×7 boards to be first player wins, and solving 6×4 to be a second player win.

D. The Game of Amazons

The game of *Amazons* was invented by Walter Zamkauskas of Argentina in 1988. It is a two-player board game played on a rectangular board, with standard size 10×10[1]. Each player controls 4 *amazons*, or *queens*, which are placed on the edge of the board before a game starts. Two players *Black* and *White* move alternately until the player to move has no legal move and thereby loses the game. *White* usually plays first. The starting position of 5×6 *Amazons* is shown in Fig. 1.

A move in *Amazons* is comprised of two compulsory phases: queen move and arrow shot. First, a player moves one queen q of

Manuscript received April 15, 2013; revised August 26, 2013; accepted February 25, 2014. Date of publication March 13, 2014; date of current version March 13, 2015. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

The authors are with the Department of Computing Science, University of Alberta, Edmonton, AB T6G 2E3 Canada (e-mail: jiaxing2@ualberta.ca; mmueller@ualberta.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2014.2309077

[1]Throughout the paper, board dimensions are presented in the format $width \times height$.

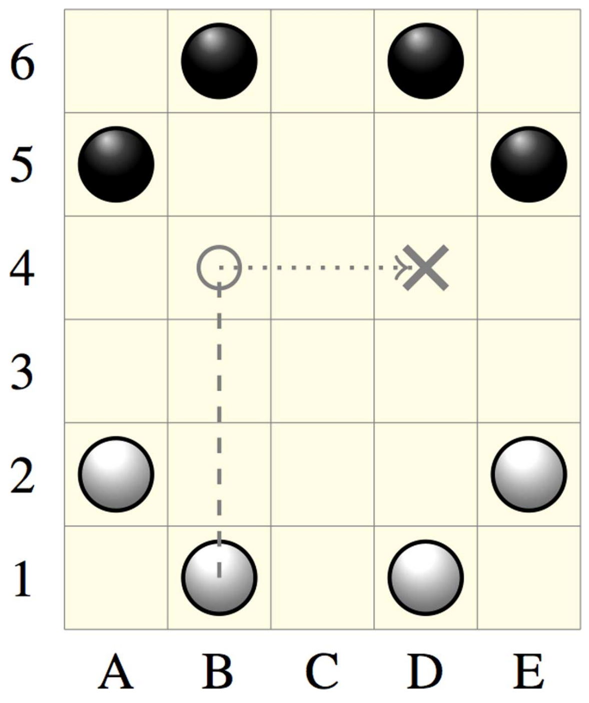


Fig. 1. 5 × 6 *Amazons* starting position and strong first move.

the player's own color from its *origin square* to a different *destination square* in a straight line either horizontally, vertically or diagonally, with the constraint that it may not cross or enter a square occupied by an amazon of either color or a burnt-off square. Next, *q* has to shoot an arrow, which travels in the exact same way as a queen, from the square *q* just moved to. The destination square of the arrow is *burnt-off* permanently from the board: no further queen moves or arrow shots can travel over or land on this square. Since exactly one empty square is burnt off in each move and at least one empty square is needed to make a move, an *Amazons* game is guaranteed to terminate in at most as many moves as there are empty squares. A strong opening move for *White* in 5 × 6 *Amazons*, shown in Fig. 1, is to move queen $B1$ to $B4$ and shoot to $D4$, abbreviated as B1-B4 × D4. The gray circle in the figure marks the destination square for the queen, and the gray cross indicates the square to be burnt off.

E. Related Work on Amazons

Amazons has simple rules, but the computational complexity of all but the most trivial *Amazons* games is high due to its large branching factor. For example, the first player has 410 moves in the 5 × 6 starting position. Even the special case of deciding whether a queen can make a certain number of moves in one-player *Amazons* puzzles is NP-complete [2], and determining the winner of a generalized *Amazons* game is PSPACE-complete [3].

Since a board often breaks down into independent subgames as the game proceeds, *Amazons* has attracted the attention of researchers in combinatorial game theory. Berlekamp analyzed positions with one queen per player on $2 \times n$ boards and determined their thermographs [4]. Tegos built databases of combinatorial game values and *defective* territories, which cannot be completely filled, and used them for building a strong *Amazons* player [5]. Snatzke also computed the canonical forms of *Amazons* positions up to size 2×11 by exhaustive search [6], and both authors identified some interesting positions.

Kloetzer, Iida and Bouzy [7], [8] solve local endgames by several different minimax search methods, and compare algorithms for playing sums of such games. Okada *et al.* [9] seem[2] to approximate a subgame by two integers.

F. State of the Art for Solving Amazons

Only a few results are known for analyzing or solving small *Amazons* boards. On boards with width or height less than four, there is no natural way to place the standard four amazons for each player. $1 \times n$ *Amazons* with one queen each is quite trivial. Let $n = b + w + s + 2$, where s is the number of "shared" empty points between the black and white amazon, and b and w the number of empty squares accessible only by black and white, respectively. If $n = 2$, then no queen can move and if $s = 0$, the game is the number $b - w$ in terms of combinatorial game theory. Otherwise, each player can move to block the opponent and claim all the $s - 1$ remaining shared points, and the game has the value $b - w \pm (s - 1)$, with mean $b - w$ and temperature $s - 1$.

Sums of $2 \times n$ positions with one queen per color per subgame were studied in detail by Berlekamp [4]. For the standard setup with four queens each, starting on the (1,2)-points in the corner, 4 × 4 *Amazons* "is a second player win, and can be solved easily by brute-force search [10]." 5 × 5 is a first player win [11].

Table I summarizes the previously known and new results about solving *Amazons* on small boards. First player wins were found for the 4 × 5, 5 × 4, 4 × 6, 5 × 6 and 4 × 7 boards, while 6 × 4 turned out to be a second player win. The narrow and high versions seem much easier to prove as first player wins, since an effective first blocking move exists. 5 × 4 was much harder than 4 × 5, and while 4 × 6 is a fairly straightforward win by blocking, 6 × 4 is a second player win, and it took almost a full CPU day of computation to refute all 256 possible starting moves.

To the authors' knowledge, no results on other board sizes, with other numbers of queens, or with other starting locations of the queens have been published.

TABLE I
WINNERS FOR *AMAZONS* ON SMALL BOARDS. 1 MEANS A FIRST PLAYER WIN, 2 A SECOND PLAYER WIN. NEW RESULTS ARE CIRCLED

width \height	4	5	6	7
4	2	①	①	①
5	①	1	①	?
6	②	?	?	?

II. BACKGROUND

A. Solving Amazons

1) Simple Bounds: The deciding factor for analyzing an *Amazons* position is the difference in the number of moves that the players can make. If this number is zero, then the last player

[2]The English summary of this Japanese paper is vague about this point.

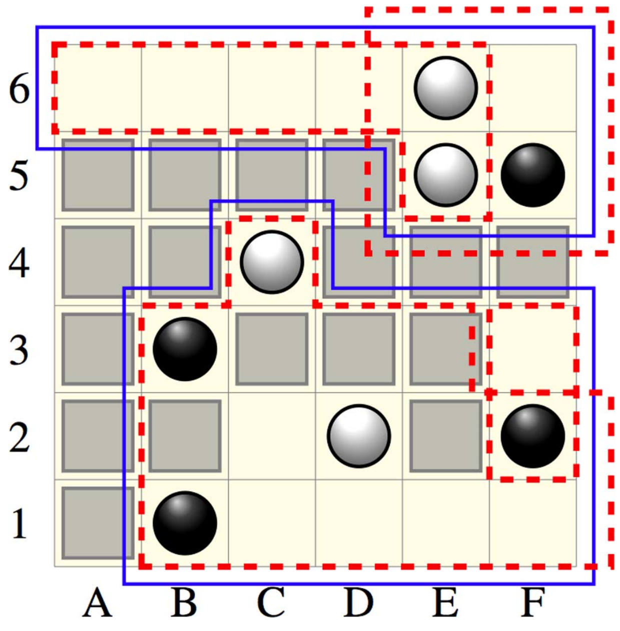


Fig. 2. Board partition.

able to move wins. Simple integer bounds of a game G, as used in [11], limit the range of the number of moves *Black* can make more than *White* in G. Simple bounds of a game are written in the form $[lower, upper]$ where $lower$ is a lower bound on this number, and $upper$ is an upper bound. Given bounds b, the notation $l(b)$, $u(b)$ is used to indicate these lower and upper bound, respectively. For example, if the bounds of a game are $[-3, -1]$, then *Black* can make at least -3 and at most -1 more moves than *White* (i.e., *White* has at least 1 and at most 3 more moves than *Black*), therefore, *White* wins.

The following rules [11] relate the winner of a game G to its simple bounds $b = bounds(G)$:

- black wins if $l(b) > 0$, or if both $l(b) = 0$ and it is White to move;
- white wins if $u(b) < 0$, or if both $u(b) = 0$ and it is Black to move;
- in other cases, bounds b do not provide enough information to determine the winner of G.

These bounds and rules will be generalized in Section II-D.

2) Amazons Solver Architecture: Since neither arrows nor queens can pass through a burnt-off square, these squares establish a barrier between different parts of the board. As the game progresses and squares are burnt off, the board naturally splits into independent subgames (or areas). For example, in Fig. 2, two such areas are delimited by solid lines.

Bounds on the values of independent areas can be computed individually and summed up into a single global bound, which can then be used to determine the winner of the whole board according to the rules above.

3) Board Partition: Intuitively, areas can be found by finding all the 8-connected components of empty squares and queens of both players on the board. This *basic partition* yields the two areas delimited by solid lines in Fig. 2.

An *improved partition* can be achieved by using queens of one color to block the opponent [11]. In the basic partition, if part of an area is inaccessible to the opponent given that the player does not move, then this part of the board along with the blocking queens (called **blockers**) constitutes a new area, called a *blocker territory*. The nonblocker queens are called *normal* queens. In the example in Fig. 2, the improved partitions are delimited by dashed lines. The *White* queens $E5$ and $E6$ and the *Black* queen $F2$ are blockers. An improved partition splits an area from the basic partition into multiple smaller areas by using blockers. A blocker belongs to all the areas it separates, to illustrate the fact that it can potentially move in any of them.

4) Solver Overview: Algorithm 1 illustrates the abstract main loop of the solver. Function `NextPosition` gets the next position to be evaluated depending on the search algorithm, such as a most promising node in df-pn. The position is partitioned and the bounds of each resulting area are computed. The computed bounds are summed up into `globalBounds`, and `UpdateSolver` checks whether the winner of this position can be determined from the generalized rules in Section II-D. The solver's internal state is updated with that information. For example, df-pn will update its proof and disproof numbers. This process is repeated until the starting position is solved or resources run out.

Algorithm 1 Abstract Solver Main Loop

```
function SolverMain(root)
    while root is not solved do
        position←Nextposition()
        areas←BoardPartition(position)
        ComputeBounds(areas)
        globalBounds←SumBounds(areas)
        UpdateSolver(globalBounds)
    end while
end function
```

B. Areas in Amazons

Different detailed analyses of the areas discussed in Section II-A2 are the main building blocks of the solver. An *area* in *Amazons* is a set of 8-connected squares which can contain empty squares and queens of either color. While areas don't contain any burnt-off squares, such squares are often shown in diagrams to improve readability. An area containing only queens, or only empty squares, is called a *dead area* [11], and is ignored in further analysis since neither player can move there.

In a basic partition, the resulting areas are all independent by construction. In an improved partition, areas can overlap on blockers. Based on whether both players have queens in an area and whether blockers exist, areas can be classified as active, simple territory or blocker territory. The *size* of an area is defined as the *(width, height)* pair of its minimum bounding rectangle.

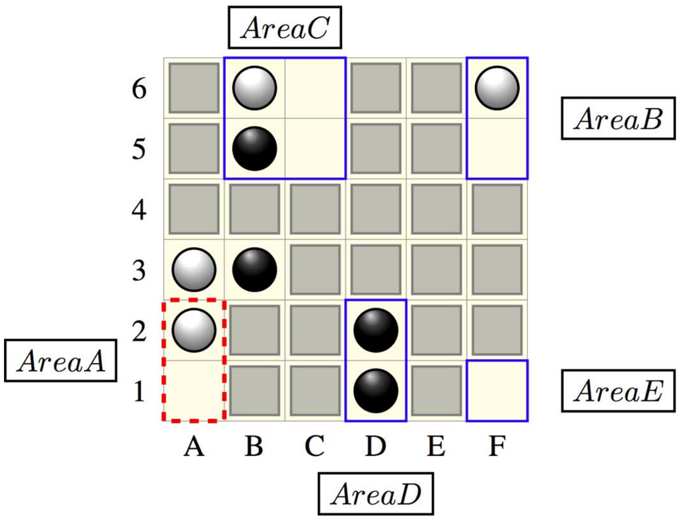


Fig. 3. Area classification example.

1) Active Area: An active area contains at least one queen of each color and one empty square. For example, *Area C* in Fig. 3 is a 2×2 active area. Playing in an active area is often good, as it can give a player the chance to block off more squares from the opponent. However, in a *zugzwang*, playing first is a disadvantage [10].

2) Simple Territory: A simple territory contains queens of only one color and does not overlap with other areas. For example, *Area B* in Fig. 3 is a 1×2 simple territory for *White*. Since only one player has moves in a simple territory, its absolute value is an integer, the maximum number of moves that the owner can make. Determining this value is not trivial, since territories can be *defective* and provide fewer moves than the number of empty squares [10]. Examples are shown in Fig. 7.

3) Blocker Territory: A blocker territory contains queens of only one color and overlaps with active areas, or with other blocker territories, on its blockers. For example, *Area A* in Fig. 3 is a *White* blocker territory of size 1×2. Like simple territories, blocker territories are one-player games with integer value when considered in isolation. However, sometimes a player can profit from moving into an adjacent area and (re-)moving the blockade. Such moves are considered by the solver in its search, but are not used for computing bounds from blocker territories. See the detailed discussion in Section IV-B3.

Having blocker territories separated provides an advantage for both playing and solving the game. From a strategic point of view, blockers provide a player the advantage of securing territory-to-be as well as threatening other parts of the board. For the solver, evaluating the blocker territory separately often yields a stronger bound that leads to a faster proof. Finally, separating out blocker territories shrinks the size and thereby simplifies remaining active areas.

C. Combinatorial Game Theory Background

Concepts from combinatorial game theory [12], [13], such as sums of games, canonical form and thermographs, can be used to help evaluate *Amazons* positions. Each area corresponds to a *subgame*, and each move affects exactly one subgame. Territories in *Amazons* correspond to integers in the theory. Many other game values occur in active areas, such as fractions, hot games, and infinitesimals. While the *canonical form* of an area contains all information needed to play perfectly in a sum of this and other areas, the average complexity of the canonical form grows very rapidly with the size of areas. In contrast, the *thermograph* of a subgame is of bounded size in practice—fewer than 20 rational numbers suffice to describe all but extremely rare cases.

1) Thermographs: A thermograph— Fig. 4(c) shows an example—is a data structure consisting of two sets of line segments, called the left and right scaffold, which describe the minimax value of a game when played with a *tax* $t \geq 0$, for Left = Black = positive and Right = White = negative going first, respectively. The scaffolds meet to form a single *mast* at the temperature of the game. In subzero thermography [14] as used here, the tax can also be negative, down to a value of -1. This extension of thermography allows stronger comparisons of games by adding detail about a subgame's behavior at low temperatures, which occur frequently on small *Amazons* boards.

The current work proposes several ways to use thermographs of subgames for computing relaxed bounds on the value of a game, and also by using infinitesimals to improve these bounds and prove that one player can make the last move. While thermographs of two games can not be added to compute the thermograph of the sum, the bounds derived from them can be added, and the information about positive or negative infinitesimal values can be combined, by using the fact that the sum of positive games is again positive, and the sum of negative games is negative. The next subsection formalizes such extended use of bounds.

D. Generalizing Bounds: Tightened and Relaxed Bounds

Combinatorial game theory defines a space of game values that is much richer than just integers [13]. Relaxed bounds use two of these concepts, fractions and *infinitesimals*, as follows:

1) Bounds can be fractions, not just integers. For example, $\text{bounds}(G) = [-3/2, 5/16]$ means that $G \geq -3/2$ and $G \leq 5/16$ in terms of combinatorial game theory.
2) Both upper and lower bounds can be independently *tightened* or *relaxed* .

A tightened bound excludes the number that defines the bound itself, and is written in standard mathematical notation using a round bracket. For example, $\text{bounds}(G) = [-3, -1)$ means that the lower bound is $G \geq 3$, while the upper bound is tightened to exclude the value -1, so $G < -1$. The opposite, a relaxed bound, means that the game value can be infinitesimally outside the bound. A relaxed bound is written by appending the symbols $+\epsilon$ or $-\epsilon$ to the number. For example, the relaxed bound $\text{bounds}(G) = [-3, -1+\epsilon]$ means that while $G < -1 + \epsilon$ for all $\epsilon > 0$, $G \leq -1$ is not proven. $-1 + *$ is an example of a game which would have such a bound. Similarly, $\text{bounds}(G) = [-3 - \epsilon, -1]$ means that while $G > -3 - \epsilon$ for all $\epsilon > 0$, $G \geq -3$ has not been proven. In contrast to simple and tightened bounds, a relaxed bound of 0 cannot be used to establish a win or loss. Both tightened and relaxed bounds can result from using thermographic databases, and more examples of such bounds can be found in Sections II-D1, II-D2, and IV-B2. Addition of bounds uses the following rules, which are applied independently to the upper and lower bounds:

1) If at least one of the bounds is relaxed, then the sum is relaxed. For example, $[1, 2+\epsilon] + [1-\epsilon, 3] = [2-\epsilon, 5+\epsilon]$ and $(1, 2) + [1-\epsilon, 3+\epsilon] = [2-\epsilon, 5+\epsilon]$.
2) Otherwise, if at least one bound is tightened, then the sum bound is also tightened. For example, $[1, 2] + (1, 3] = (2, 5]$ and $(1, 2) + [1-\epsilon, 3] = [2-\epsilon, 5)$.
3) Otherwise, the bound includes the number and is not relaxed. For example, $[1, 2] + [1, 3] = [2, 5]$.

The following rules can determine the winner of a game G from its relaxed bounds $b = \text{bounds}(G)$:

- *Black* wins if:
  - $l(b) > 0$;
  - $l(b) = 0$ and the lower bound has been tightened;
  - $l(b) = 0$, the lower bound is not relaxed, and it is *White* to move.
- Similarly, *White* wins if:
  - $u(b) < 0$;
  - $u(b) = 0$ and the upper bound has been tightened;
  - $u(b) = 0$, the upper bound is not relaxed, and it is *Black* to move.

In all other cases, bounds b do not provide enough information to determine the winner of G. However, in Section IV-C2, extra rules are developed which look at some of the components of a sum game in order to recognize more wins.

While the notion of relaxed and tightened bounds is similar to the concept of *confusion interval* in combinatorial games [13], there are three distinct cases for each bound in the current framework.

1) Relaxed Bounds From Thermographs for Hot Games: Hot games provide extra moves for the player who makes the first move locally. The *left and right stops*, the values of the left and right scaffolds at temperature $t = 0$, can be used as bounds on the value of a game as follows: If the slope of the scaffold is vertical below $t = 0$, then the bound is exact. If it is diagonal, then the game is *confused* with this value, and the bound must be relaxed.

For example, Fig. 4 shows an active area a and its thermograph, with a left stop of 2 and right stop of 1 computed at temperature 0. The lower bound of 1 is exact since the thermograph is vertical at $t = 0$, but the upper bound of 2 is relaxed, since the slope is diagonal there, so $\text{bounds}(a) = [1, 2+\epsilon]$.

2) Thermograph Bounds for Infinitesimal Games: In infinitesimal games, no player can gain additional moves, but in close games they can be very important for determining who can make the last move. Infinitesimals can be grouped into four types depending on which player gets to make the last move. In positive games, Black gets the last move no matter who goes first, while in negative games, White does. Games of value 0 are second player wins, while all other infinitesimals are fuzzy, meaning a first-player win locally. An important property of subzero thermography is that the type (but not the exact value except in the case of 0) of an infinitesimal is preserved in its thermograph. Fig. 5 shows these graphs.

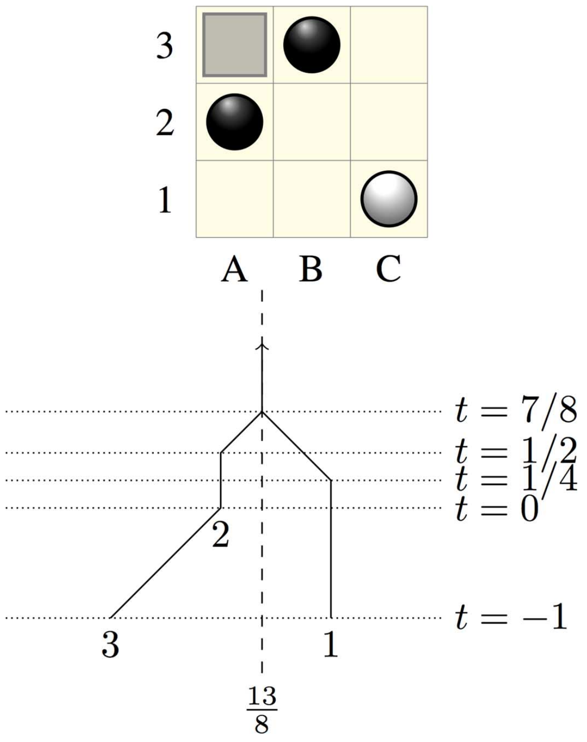


Fig. 4. Active area and its thermograph, with bounds $[1, 2+\epsilon]$.

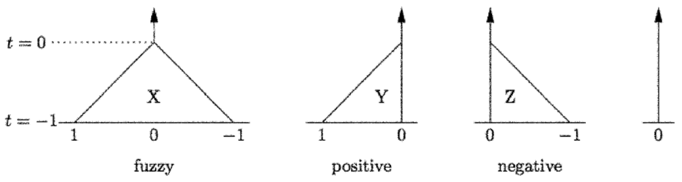


Fig. 5. The four thermographs of loopfree infinitesimals [14, Fig. 17].

The corresponding relaxed bounds on an infinitesimal G are $[0, 0]$ for $G = 0$, $(0, \epsilon]$ for $G > 0$, $[-\epsilon, 0)$ for $G < 0$ and $[-\epsilon, +\epsilon]$ for fuzzy G.

The simplest fuzzy infinitesimal is $* = \{0|0\}$, the game where either player can move first and move exactly once, ending the game. Area A in Fig. 11 is an example. $*$ is an idempotent under addition, meaning that $* + * = 0$. This is exploited in the solver by recognizing games of value $*$ and dealing with them separately from the rest of the sum, by simply removing pairs of $*$ from the sum. Other fuzzy infinitesimals are also counted separately since a single fuzzy infinitesimal is a first player win. Details on how these facts are used for improved winner detection are given in Section IV-C2.

III. Endgame Databases in *Amazons*

In board games in general, endgame databases can be either *full-board* or *partial-board* depending on what information is stored. A full-board endgame database contains precomputed exact values of game positions. For a *convergent game*, where the number of possible full-board states decreases as the

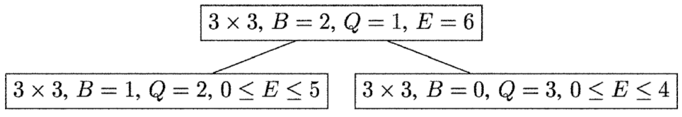

3×3, $B = 2$, $Q = 1$, $E = 6$
3×3, $B = 1$, $Q = 2$, $0 \leq E \leq 5$
3×3, $B = 0$, $Q = 3$, $0 \leq E \leq 4$

Fig. 6. Blocker territory database dependency example.

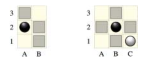


Fig. 7. Territory Bounds: Left: Simple territory, bounds $[1, 2]$. Right: Blocker territory, bounds $[1, 3]$.

game progresses, such databases reduce the depth of the search. Large endgame databases have been successfully used to solve checkers, Nine Men's Morris, Awari and many other games [15].

For *divergent games* such as *Amazons* or Go, it makes no sense to build such full-board databases. However, if the game board can be split into independent subgames, then partial-board endgame databases can provide perfect information for at least some subgames, which helps the overall solving process. An advantage of partial-board databases is that they can be used regardless of the full board size, as in Tegos' *Amazons*-playing program *Antiope* [5].

Amazons endgame databases are built for the three different area types of Section II-B. This section focuses on describing the new blocker territory databases, since the other two types have been built and used before.

A. Building Blocker Territory Databases

The computational technique used to generate the blocker territory databases is *retrograde analysis*. This process starts by generating terminal positions whose values are known statically, then computes positions that lead to a terminal position directly by doing a 1-ply lookup. Next, all positions 2 ply away from terminal positions can be solved by a 1-ply lookup, and so on until either some stopping criteria such as the memory limit, or the beginning of the game is reached.

Retrograde analysis for blocker territory databases is complicated by the special behavior of blockers. When a blocker moves into a blocker territory, it must shoot back to its origin square in order to prevent the opponent from getting into this territory. This property is called the *blocker constraint*. After this initial move, a blocker turns into a normal queen which can move and shoot freely. Therefore the values in a blocker territory database depend on positions where some of the blockers have become normal queens: the positions in a blocker territory database of size $w \times h$ with B blockers, Q normal queens and $E \leq w \times h - B - Q$ empty squares potentially depend on positions of the same size with b ($0 \leq b \leq B$) blockers, $Q + B - b$ normal queens and e ($0 \leq e \leq E - B + b$) empty squares. For example, Fig. 6 shows that the 3×3 blocker territory database with 2 blockers, 1 normal queen and 6 empty squares depends on 3×3 positions with 1 blocker, 2 normal queens and up to 5 empty squares, as well as on 3×3 positions with 0 blockers, 3 normal queens and up to 4 empty squares. Note that each move by a blocker also burns off a square and thereby reduces the number of empty points left.

After the databases covering the dependencies are built, retrograde analysis can be applied in the usual way.

B. Databases Used

1) Simple Territory Databases: The simple territory databases are similar to those of Tegos [5]. However, the blocker territory database entry format was used for the simple territory databases rather than *line segment graphs* as in Tegos' implementation. 48 simple territory databases were computed: $1 \times n$, $2 \times n$ and $3 \times n$ databases for all $2 \leq n \leq 6$, except for 3×6 with 4 queens. The database statistics are summarized in [16, Table B.1].

2) Active Area Databases: Active area databases use the implementation by Enzenberger which is part of the code base for the *Amazons* program Arrow [17]. Each entry stores the position as in the territory databases and a reference to its corresponding thermograph [16]. The active area databases are built without blocker constraints. Therefore, they cannot be used to query a remaining active area when blocker territories are partitioned out in an improved partition. All nontrivial $1 \times n$ and $2 \times n$ databases were built for $2 \leq n \leq 6$, plus $3 \times n$ for $3 \leq n \leq 5$, 3×6 with at most two queens each, and the four smallest 4×4 databases. [16, Table B.2] shows their statistics.

3) Blocker Territory Databases: Similarly, 106 blocker territory databases were built and summarized in [16, Table B.3].

IV. Improving Area and Global Bounds

This chapter discusses how the bounds of different types of areas are computed. Section IV-A shows how bounds of each type of area are initialized by using heuristics, or estimated by local search. Section IV-B describes how tighter bounds can be obtained from databases. Section IV-C describes how local bounds are combined to try to find a winner for a given full board position.

A. Computing Bounds by Heuristics

1) Simple Territory: Since there is no easy way in general to determine whether a simple territory can be completely filled or not (see Section II-B), as in [11] a simple *plodding* heuristic is used to quickly compute a basic lower bound p. Then the bounds of a simple territory with v empty squares are initialized to $[p, v]$ for *Black*, and $[-v, -p]$ for *White*. The heuristic counts the number of moves each queen can make by one sampled sequence of *plodding moves*, which move to a neighboring empty square and shoot back. Each visited empty square is marked and not used by other queens. For example, the bounds of the position shown on the left of Fig. 7 are $[1, 2]$ since the heuristic can only make 1 move with this *Black* queen. In this defective territory, 1 is the true number of moves, but in general the heuristic just returns a lower bound.

2) Blocker Territory: The bounds of a blocker territory are initialized in the same way as a simple territory. When filling a blocker territory, the plodding heuristic starts by using normal queens, and uses blockers only when necessary.

For example, the position shown on the right in Fig. 7 is a *Black* blocker territory with blocker $B2$. The bounds computed by the heuristic are $[1, 3]$ since the blocker has to shoot

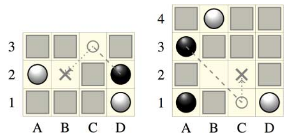


Fig. 8. New static evaluation Rules 5 and 6 for queens with 1 and 2 AES.

back when it moves, thus blocking its access to the other empty squares.

3) Active Area: The bounds of an active area with v empty squares are initialized to $[-v, v]$, then improved by either static evaluation or local $\alpha\beta$ search. Static evaluation improves bounds by identifying safe moves for the players. This is often only a small improvement, but is fast enough to compute for all active areas. Local search computes bounds for an active area by exhaustive $\alpha\beta$ search. For performance reasons, it is only used for relatively small nondatabase positions.

4) Safe Moves: The concept of *safe moves* was introduced in [11] in order to improve bounds on an *Amazons* area. Safe moves are guaranteed for a player, since the opponent cannot eliminate them. Several examples will be shown below.

5) Static Evaluation: A queen needs at least one *adjacent empty square* (AES) to make a move. The purpose of static evaluation is to find out if there are safe moves for either player. Bounds are improved according to the following rules:

1) For every *Black* safe move, the lower bound is increased by 2;
2) For every *White* safe move, the upper bound is decreased by 2.

Each safe move changes a bound by 2 since one safe move for a player also means one less potential move for the opponent. For example, if *Black* has two safe moves and *White* has one safe move in an active area with v empty points, then the improved bounds are $[-v+4, v-2]$.

How many safe moves a queen can make depends on its AES status, the opponent's queen distribution and whose turn it is to move next. Four rules for finding safe moves were defined in [11], and are also implemented in this solver. The current work contributes two new static evaluation rules, Rule 5 and Rule 6, as follows:

Rule 5 takes the origin square of the opponent queen which blocks a single AES into account. For example, in Fig. 8 on the left, the *White* queen $A2$ has 1 AES $B2$ which the *Black* queen $D2$ can block by D2-C3× B2 as shown in the figure. However, in blocking this AES, *Black* has to free the *White* queen $D1$, so one safe move can be claimed for the two *White* queens combined.

Rule 6 is similar to Rule 5 for a queen with 2 AES. In Fig. 8 on the right, the *White* queen $D1$ has 2 AES and the *Black* queen $A3$ can block both by moving to $C1$ and shooting to $C2$. However, this opens up $A3$ for *White* queen $B4$, and *White* can claim one safe move for $D1$ and $B4$ combined.

There are potential dependencies amongst safe moves for different queens [11]. Therefore, static evaluation is only applied at most once for each player in an active area.

6) Local Search: Bounds of an active area can be computed on the fly by local $\alpha\beta$ search. Two such searches are done for each area, one for each player going first. The search score is the difference in number of moves on the board between the first player and the opponent. Passes are allowed for both players even if they have other legal moves, in order to handle zugzwang positions correctly. A search ends after two consecutive passes or when there are no more empty squares.

Let the local search value from the first player's point of view be v_w and v_b for *White* and *Black* moving first in an active area A, respectively. Then the bounds of A are set to $[v_w - \epsilon, v_b + \epsilon]$. For efficiency, local searches are only performed on nondatabase positions with total number of (queens plus empty squares) less than 9.

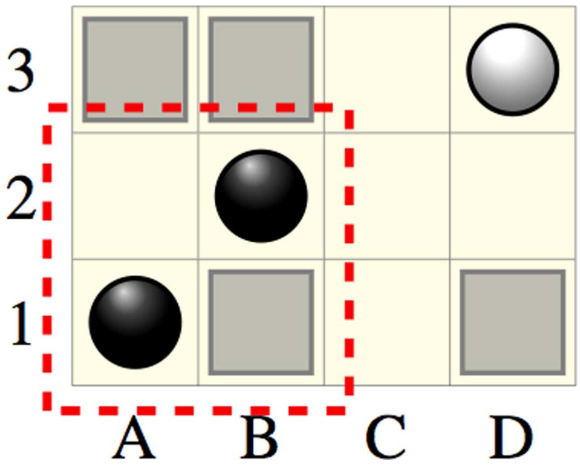


Fig. 9. Blocker territory with 1 normal queen and 1 blocker.

B. Computing Bounds From Databases

1) Simple Territory Databases: When a simple territory is encountered during a search, it is queried against the simple territory database with the corresponding size and number of queens. If the query is successful, its exact value v is returned and the bounds for this territory are set to $[v, v]$. As an example, for the defective territory in Fig. 7 on the left, the query returns the exact value of 1 and the bounds are set to $[1, 1]$.

2) Active Area Databases: An active area encountered during the search is queried against the active area database with the corresponding size and number of queens of each color. If the query succeeds, the thermograph of this area is retrieved and (possibly relaxed) bounds for the area are computed as in Section II-D1.

3) Blocker Territory Databases: Blocker territories are the result of an improved partition of an active area. In the case that the original active area is not in the database as a whole, an improved partition is computed. If no blocker territories can be partitioned out, then the bounds of this active area are computed as in Section IV-A3. If the improved partition succeeds, then the bounds of the resulting blocker territories and active areas are computed separately. A blocker can be used to improve the bounds of at most one area that it is part of. For example, Fig. 9 shows a blocker territory delimited by the dashed line, containing one normal queen $A1$ and one blocker $B2$. This blocker can either be used to help fill the blocker territory, or to move in the remaining active area.

In the current implementation, neither active area databases nor local searches consider the blocker constraint. Bounds on the remaining active areas are only computed with static evaluation as in Section IV-A3, with the additional constraint that blockers that are used in other areas must not be used to find safe moves (either by themselves or combined with other queens). The more blockers can be spared from use in blocker territories, the better the chances for achieving tighter active area bounds. For example, in Fig. 9, if the blocker $B2$ is used for filling the blocker territory, then it cannot be used to improve the bounds of the remaining active area r. The bounds of r are therefore $[-4, 2]$ since *White* has a safe move. If queen $A1$ is used to fill $A2$, then the blocker $B2$ can be used to improve the bounds of r to $[-2, 2]$ because it has a safe move in r.

Unfortunately, in order to reduce the size of blocker territory databases, the current implementation does not store the information about which queens are used for filling a territory. Therefore, a query needs to assume that all blockers and normal queens are used for filling, even if the territory can be filled with only a small subset of them, like queen $A1$ in the example above.

4) Changing the Query Position: To ameliorate this problem, three heuristics select a subset of all the queens in a blocker territory B, to create a new query position B' which reduces the number of blockers used. The remaining blockers are not chosen for filling this territory, and are free to be used for other adjacent areas. One drawback of eliminating queens from the query position B' of a blocker territory B is that if B' is defective, it is not sure whether B is defective or not. In terms of bounds, bounds$(B) = [u(bounds(B')), e]$, where e is the number of empty squares in B. If B' is nondefective, then B is also shown to be nondefective since it can be filled completely. For example, if the square $C2$ containing a blocker is removed from the blocker territory in Fig. 10, then this smaller territory within the solid line becomes defective, so the computed bounds are $[1, 2]$. Details of the queen selection process are described in[16, Ch. 4.2.3].

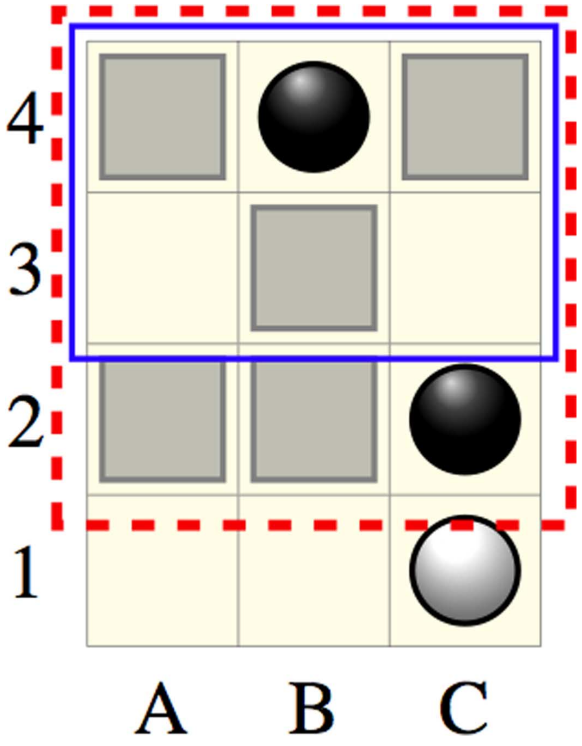


Fig. 10. Territory becomes defective without the blocker $C2$.

C. Combining Bounds

After a game position G is partitioned into areas, the bounds of each area are computed and added to yield a single global bound. This bound is then checked against the rules in Section II-D to try to determine the winner of G. For *simple territories* computed by heuristics or from the databases, *blocker territories* computed by heuristics or from the databases, assuming blockers blocking multiple blocker territories are handled properly as stated in Section IV-B3, and for *active areas* computed by *static evaluation and local search*, the combination is just bounds summation. Combining *active areas computed from thermographs* is more complex since more detailed information about these games in their thermographs can be used to achieve better bounds. The handling of these games is explained in the next two subsections.

1) Handling Hot Game Bounds From Thermographs: Bounds on a sum game can be improved by simulating a one move lookahead. If at least one hot game is present, the best improvement is achieved by assuming that the next player will move in a hot game with *widest bounds* b_w, which are bounds for which the difference $u(b_w) - l(b_w)$ is largest. Then in the sum, b_w can be improved as follows: If Black is to play, raise the lower bound $l(b_w)$ (which could also be a relaxed bound of $l(b_w) - \epsilon$ or tightened) to $u(b_w) - \epsilon$. Analogously, if White is to play, lower the upper bound from $u(b_w)$ to $l(b_w) + \epsilon$. For example, if the widest bounds in the sum are $[-5, 2]$, they can be replaced by $[2 - \epsilon, 2]$ if it is *Black*'s turn, and by $[-5, -5 + \epsilon]$ if *White* plays next.

This replacement yields valid global bounds since it corresponds a conservative estimate of the effect of the first player moving in this subgame and committing to keep replying to opponent's moves here until the stop value is reached. Because of the minimax principle, a bound computed from restricting the first player's strategy is a lower bound (from the player's point of view) on the best result that player can achieve. Using the next player's privilege of moving first in this way precludes using it again for infinitesimals, as in the next section.

2) Handling Infinitesimals: In principle, bounds for infinitesimals can be handled just like other games, according to the addition rules inSection II-D. However, games of value $*$ and other fuzzy infinitesimals are collected separately in the solver, to take advantage of the fact that $* + * = 0$ and to identify some more wins as explained below. If the next move privilege was not used on a hot game, it can potentially be used with an infinitesimal to prove more wins. The rules below are for *Black*, with the rules for *White* obtained easily by negating all values and outcomes. Let b be the bounds on a sum game excluding infinitesimals, with $l(b) = 0$, and the lower bound not relaxed. Then *Black* as the next player wins if:

- no other infinitesimals exist and there is an odd number of $*$s;
- no other infinitesimals exist and there is a single fuzzy infinitesimal;

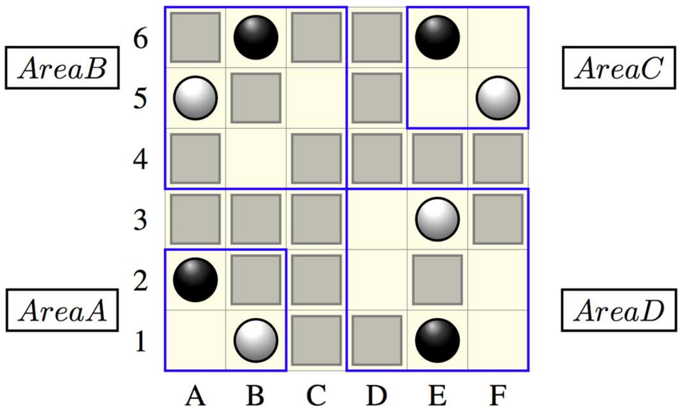


Fig. 11. *Black* to move can win, value $*$.

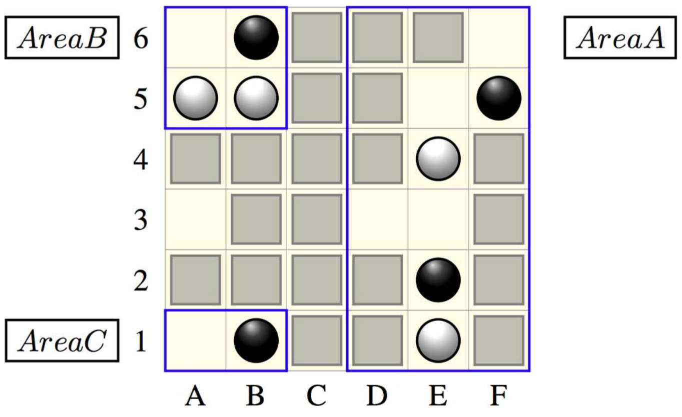


Fig. 12. *White* to move, global bounds $[1-\epsilon, 3+\epsilon]$.

- all other infinitesimals are positive and there is either an odd number of $*$, or a single fuzzy infinitesimal, but not both.

In all these cases *Black* wins by playing first in the fuzzy infinitesimal (which might be a $*$) and leaving a game $G^L \geq 0$ for *White*.

3) Board Evaluation Examples: A Win by $*$: Fig. 11 shows a 6×6 *Amazons* endgame position with 9 empty squares left. The board is partitioned into 4 active areas and all the areas can be looked up in the databases. Area A is recognized as a $*$, and areas B, C, and D are all the integer 0. Therefore, the sum game is evaluated as $0 + *$ and the first player, *Black* in this case, can win by moving in the $*$, leaving 0 for *White*.

Without databases, area A would still be recognized as a $*$, but B, C, and D would be searched locally, resulting in each of them bounded by $[-\epsilon, +\epsilon]$. The exact nature of these infinitesimals would remain unknown, and the winner could not be determined without search.

A *Local Search Win*: Fig. 12 shows a 6×6 *Amazons* endgame position with 6 empty squares left. Area B is recognized as a $*$. Area C is a simple territory of value 1. Area A, of size of 3 $\times$ 6 with two queens each, is not in the current databases. Local search results in bounds $[-\epsilon, 2+\epsilon]$. The global bounds (both with or without the $*$) are $[-\epsilon, 2+\epsilon] + 1 = [1-\epsilon, 3+\epsilon]$, securing a win for Black.

Without local search, Area A has bounds $[-2, 2]$ since both players have one safe move there. The global bounds become $[-2, 2] + 1 + * = [-1, 3] + *$, and no winner can be determined from these bounds.

V. Experiments and Results

A. Solving Test Cases

Two *Amazons* solvers were developed within the Arrow framework:*ab* is a $\alpha\beta$-based solver similar to the one used in [11], and *df-pn* is the new df-pn based solver. In the first experiment, the performance of the solvers and databases was compared using the same test data as in [11], which contains positions selected from three test games named *f1*, *f2* and *f3*. *f1* and *f2* are 5×5 games and *f3* was played on 6×6. The test set is available at [18]. The configuration of the solvers is as follows: *ab* uses a hash table with 2^{25} entries and no databases, and *df-pn* has a hash table with 2^{22} entries. This is roughly fair in terms of memory usage since individual table entries are larger. *df-pn* is evaluated in all 8 combinations of including/excluding the three types of databases.

The search results for *f1*, *f2* and *f3* are shown in Fig. 13. In the largest test cases, *ab* exceeded the time limit of 10000 seconds per search. For each graph, the horizontal axis is labelled with the test case number for even-numbered test cases. The number of empty squares in the test cases increases from left to right. The vertical axis shows the number of nodes searched to solve each case on a logarithmic scale. *df-pn* outperforms *ab*, sometimes by more than an order of magnitude, in all large test cases and most small ones. Using the databases sometimes reduces the number of nodes by another order of magnitude. Most *df-pn* results are clustered into two groups, with the versions including the combinatorial game databases outperforming the ones without. In contrast, both types of territory databases yield only small improvements. This is due in part to the small size of the board, which does not allow many large territories to be formed. Most very small territories are already evaluated correctly by the nondatabase static evaluation.

B. Solving 5 × 6 Amazons: A First Player Win

The *df-pn* solver with a hash table of 2^{30} entries plus the databases as specified in Section III-B weakly solved the 5 $\times$ 6 starting position shown in Fig. 1 as a first player win. The first move was fixed as White B1-B4xD4, and all 157 possible replies for Black were refuted, running them as separate search tasks but with a shared hash table on one Quad-Core AMD Opteron(tm) Processor 8384 running at 2.7 GHz. Out of 157 moves, 9 were solved instantly by a 1 ply search through a transposition table lookup. For example, after the Black move 2. E5-C5xC4 is refuted by a line starting with White 3. D1-D3xC2, the Black move 2. E5-C5xC2 can be immediately refuted by White 3. D1-D3xC4. Both lines lead to the same position after three moves, swapping who plays the arrow shots to C2 and C4. A total of 47 Black moves was refuted in under 1 second each. In contrast, the hardest moves 2. E5-E3xE6 and E5-E3xE5 took between three and four hours each to refute. A total of 12 Black moves took over 90 minutes each. A complete strategy for winning 5×6 *Amazons* was computed and verified with a proof tree checking program. A typical difficult line in the proof is shown in Fig. 14. For the entire line, there is only

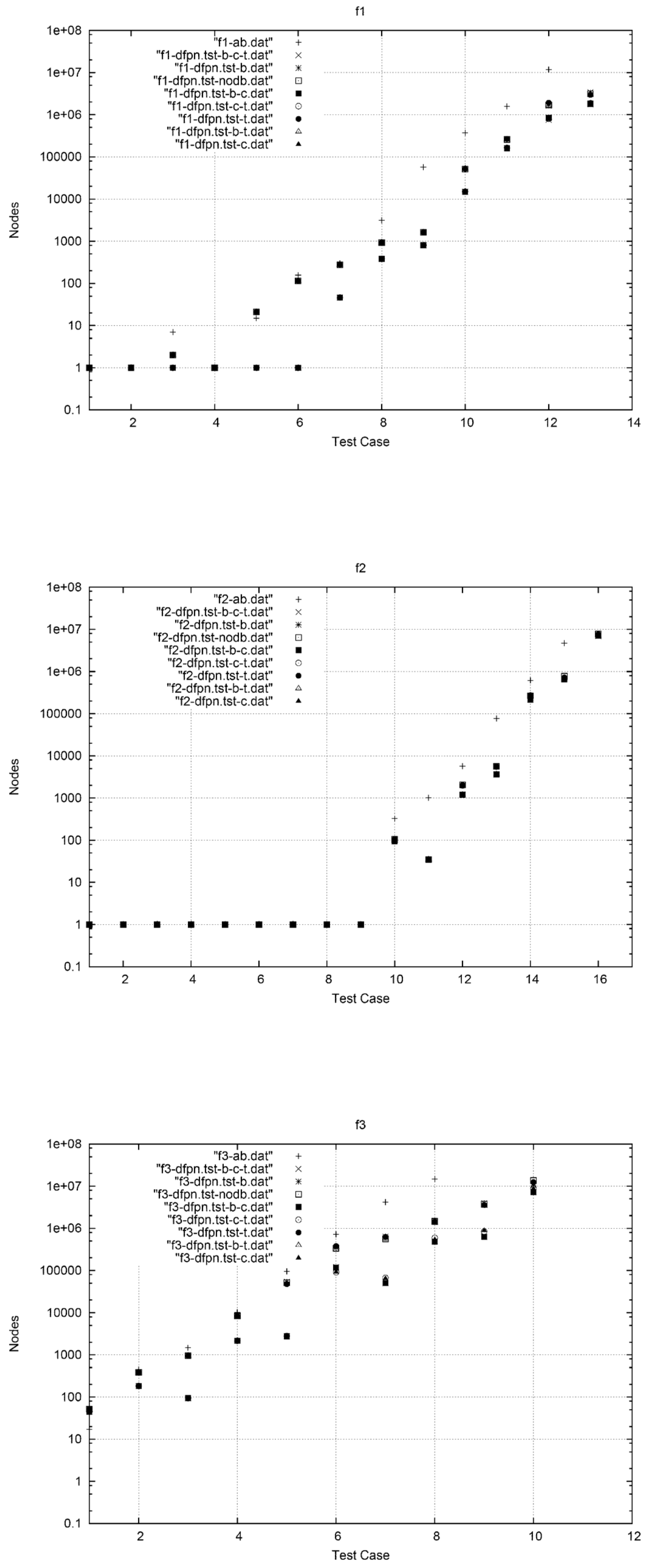


Fig. 13. Search results of *f1* (top), *f2* (middle), *f3* (bottom).

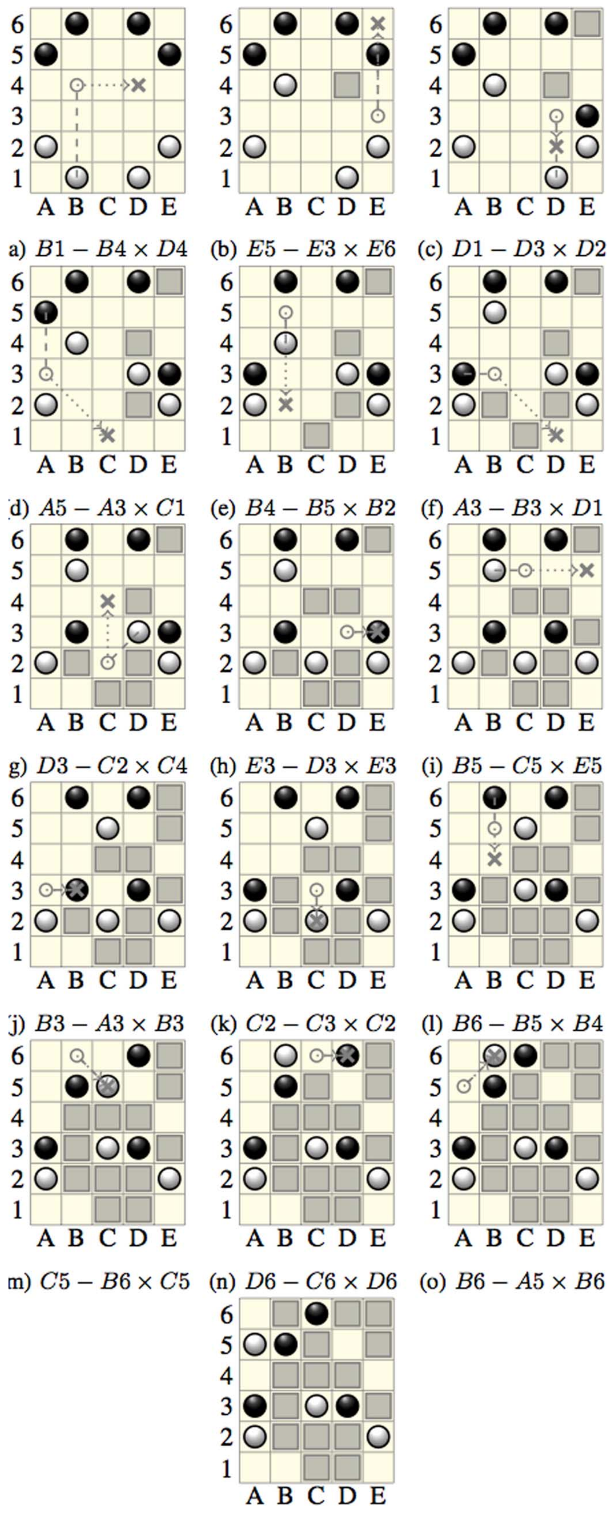


Fig. 14. A sample deep line in the 5×6 proof,*White* moves first and wins.

one active area of size 5×6. In the final position, shown at the bottom of Fig. 14, *White* has two blocker territories of value -1 and -2, respectively, while *Black* has one blocker territory of value 2. The remaining active area in the top left corner is evaluated as 0 since both players can find a safe move. The sum of these subgames is -1, a *White* win.

C. *Solving Early 6 × 5 and 6 × 6 Positions*

To further evaluate the performance of static evaluation, databases and the solver on harder and larger boards, a set of representative 6×5 and 6×6 positions was created. For both

TABLE II
SOLVING 6 × 5 POSITIONS FROM 100 GAMES. COLUMNS SHOW NUMBER OF POSITIONS SOLVED STATICALLY WITHOUT DATABASES, STATICALLY WITH DATABASES, AND WITH 500 SECONDS OF DF-PN SEARCH. RESULTS WITH ALL POSITIONS SOLVED IN BOLD

Move Nr	no DB	DB	search	total games
22	**74**	**74**	**74**	74
21	**88**	**88**	**88**	88
20	82	94	**98**	98
19	84	95	**100**	100
18	38	84	**100**	100
17	56	76	**100**	100
16	21	49	**100**	100
15	31	53	**100**	100
14	13	28	**100**	100
13	13	26	**100**	100
12	10	22	**100**	100
11	9	11	**100**	100
10	0	4	**100**	100
9	1	1	**100**	100
8	0	1	96	100
7	0	0	91	100
6	0	0	53	100
5	0	0	36	100

board sizes, 100 self-play games were played by Arrow2. These test sets are available on [18]. Starting from the end of each game, positions were tested with the static evaluation without databases, the static evaluation using all databases, and a 500 second *df-pn* search using all databases and a 2^{22} entry hash table.

Tables II and III show the results on 6 × 5 and 6 × 6 boards, respectively. On 6 × 5, static evaluation without databases can solve most positions for moves 19–22, a significant fraction of positions from move 15–18, and a small number of positions for moves 11–14, before dropping to almost zero for earlier positions. With databases, wins are recognized about 2–3 moves earlier, which is a very significant improvement given the exponential growth of the search tree. With search, 91 of 100 games on 6 × 5 are solved after 7 moves, then performance drops quickly.

On 6 × 6, the picture is similar but the decline of the two static evaluators is a bit more gradual. 72 of the games are solved by search after 8 moves. The static evaluators in both sets of games exhibit a bit of an odd/even effect, since many of these positions are wins for White, and with White to play the extra knowledge about first player wins applies.

D. Types of Areas in Search

Fig. 15 illustrates how frequently different types of areas occur during search on a 5 × 6 board. Statistics are averaged over 100 self-play games by the Arrow-2 program, with 500 seconds time limit per game. The first diagram shows a summary of all areas followed by a separate diagram with details of the active areas. There is a sharp transition between moves 10 and 11, from a single full-board active area to positions where almost all areas are found in the database.

TABLE III
SOLVING 6 × 6 POSITIONS FROM 100 GAMES

Move Nr	no DB	DB	search	total games
28	**24**	**24**	**24**	24
27	**61**	**61**	**61**	61
26	74	75	**76**	76
25	92	**94**	**94**	94
24	98	**99**	**99**	99
23	93	99	**100**	100
22	94	96	**100**	100
21	75	90	**100**	100
20	73	84	**100**	100
19	65	79	**100**	100
18	55	72	**100**	100
17	37	55	**100**	100
16	28	40	**100**	100
15	13	25	**100**	100
14	5	20	**100**	100
13	1	8	**100**	100
12	0	12	**100**	100
11	0	0	99	100
10	0	10	96	100
9			83	100
8			72	100
7			24	100
6			16	100

VI. FUTURE WORK

A. Database Improvements

Currently the territory databases store both defective and nondefective positions since the current search engines require knowing a best move for each position. Instead, the solver could be modified to play out the integer, and possibly other, parts of the game as an "abstract game" instead of on the real *Amazons* board. This would allow to greatly compress the territory databases by pruning all nondefective positions and removing the best-move field from the data.

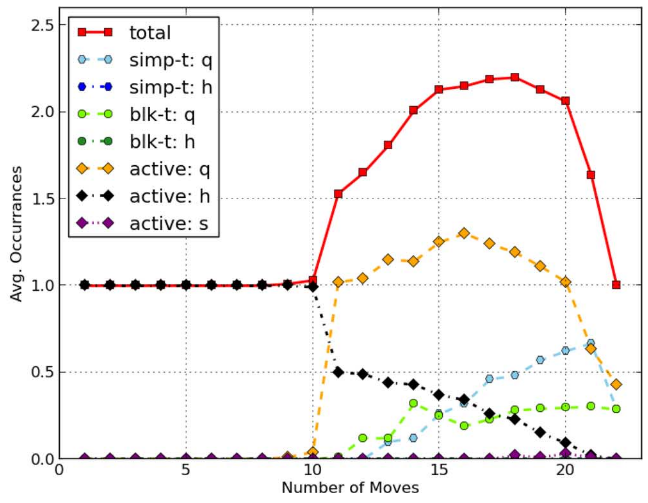
Active area databases could be built with the blocker constraint such that the remaining active areas in an improved partition can also be queried, and blocker databases could be augmented to identify the blocking queen(s).

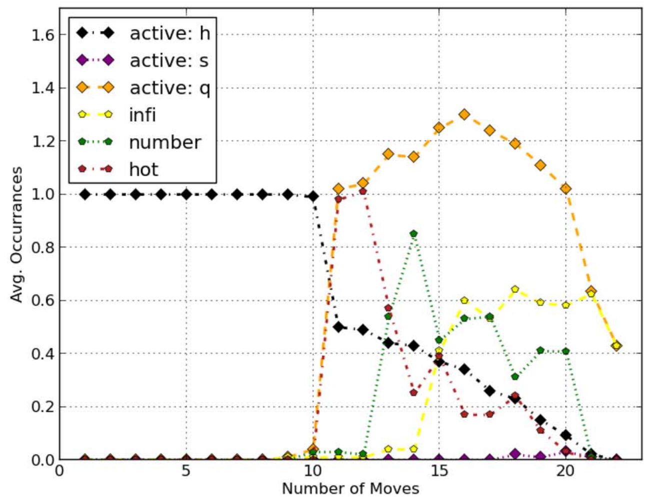
B. Parallel Computing

The solvers described above are single-threaded. For solving larger problems such as 6 × 6, it makes more sense to have a multithreaded/distributed solver with a master process/machine generating unsolved nodes and a farm of slave processes/machines actually solving them [16].

C. Search Improvements

- **Local search improvements**. Currently, the local search module does not use the databases and does not consider the blocker constraint. Both techniques should speed up these searches.





Legend	Meaning
total	the sum of all types of areas
simp-t: q	simple territories queried from the databases
simp-t: h	simple territories computed by heuristics
blk-t: q	blocker territories queried from the databases
blk-t: h	blocker territories computed by heuristics
active: q	active areas queried from the databases
active: h	active areas computed by heuristics
active: s	active areas computed by local searches
infi	infinitesimals in active: q
number	numbers in active: q
hot	hot games in active: q

Fig. 15. Area frequency on 5×6 board. Top: all types. Below: active areas, details.

- **Proof number initialization.** The proof and disproof numbers of a newly created node can be initialized to indicate how easy or hard it is expected to solve [19]. This could direct the solver towards easier to prove subtrees.
- **Area caching**. Caching search results of areas that are not in the databases but arise frequently in the search could give the solver a further performance boost.

REFERENCES

[1] J. Kloetzer, H. Iida, and B. Bouzy, “The Monte-Carlo approach in *Amazons*,” in *Proc. Comput. Games Workshop*, 2007, pp. 185–192.
[2] M. Buro, “Simple amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs,” in *Proc. Comput. Games Conf.*, 2000, pp. 250–261.
[3] T. Furtak, M. Kiyomi, T. Uno, and M. Buro, “Generalized amazons is PSPACE-complete,” in *Proc. IJCAI*, 2005, pp. 132–137.
[4] E. Berlekamp, “Sums of N X 2 Amazons,” *Inst. Math. Statist. Lecture Notes, ser. Monograph Ser.*, no. 35, pp. 1–34, 2000.
[5] T. Tegos, “Shooting the last arrow,” Master's thesis, Univ. Alberta, Edmonton, AB, Canada, 2002.
[6] R. Snatzke, “New results of exhaustive search in the game *Amazons*,” *Theoret. Comp. Sci.*, vol. 313, no. 3, pp. 499–509, 2004.
[7] J. Kloetzer, H. Iida, and B. Bouzy, “Playing *Amazons* endgames,” *ICGA J.*, vol. 32, no. 3, pp. 140–148, 2009.
[8] J. Kloetzer, H. Iida, and B. Bouzy, “A comparative study of solvers in *Amazons* endgames,” in *CIG*, P. Hiroyuki and L. Barone, Eds. New York, NY, USA: IEEE, 2008, pp. 378–384.
[9] Y. Okada, H. Kuroda, and Y. Kanada, T. Ito and T. Nakamura, Eds., “Subgame database for *Amazons*,” in *Proc. 8th Game Program. Workshop*, Japan, 2003, pp. 82–89.
[10] M. Müller and T. Tegos, “Experiments in computer *Amazons*,” in *More Games of No Chance*. Cambridge, U.K.: Cambridge Univ. Press, 2002, pp. 243–260.
[11] M. Müller, “Solving 5×5 *Amazons*,” in *Proc. 6th Game Program. Workshop (GPW 2001)*, 2001, pp. 64–71.
[12] J. Conway, *On Numbers and Games*. New York, NY, USA: Academic, 2000.
[13] E. Berlekamp, J. Conway, and R. Guy, *Winning Ways for Your Mathematical Plays*. New York, NY, USA: Academic, 1982.
[14] E. Berlekamp, “The economist's view of combinatorial games,” in *Games of No Chance: Combinatorial Games at MSRI*. Cambridge, U.K.: Cambridge Univ. Press, 1996, pp. 365–405.
[15] H. van den Herik, J. Uiterwijk, and J. van Rijswijck, “Games solved: now and in the future,” *Artif. Intell.*, vol. 134, no. 1-2, pp. 277–311, Jan. 2002.
[16] J. Song, “An enhanced solver for the game of *Amazons*,” Master's, Univ. Alberta, Edmonton, AB, Canada, 2012.
[17] M. Müller, *The Amazons-playing Program Arrow2*, 2012 [Online]. Available: http://webdocs.cs.ualberta.ca/~mmueller/amazons/arrow.html
[18] M. Müller, *Amazons test games*, 2013 [Online]. Available: http://webdocs.cs.ualberta.ca/~mmueller/amazons/games.html
[19] A. Nagai, “Df-pn Algorithm for searching AND/OR trees and its applications,” Ph. D., Dep. Inf. Sci., Univ. Tokyo, Tokyo, 2002.

Jiaxing Song, author photograph and biography not available at time of publication.

Martin Müller received the Ph.D. degree from ETH Zürich, Switzerland.

He is currently a Professor in the Department of Computing Science at the University of Alberta, Edmonton, AB< Canada. His research interests include computer game-playing and fully automated planning. He leads the open source project Fuego, which has been used to develop programs for many games including Go, Amazons, and Hex. In 2009, the Fuego Go program was the first to beat a top level Professional human player on even terms in a 9x9 game.