TDS+: IMPROVING TEMPERATURE DISCOVERY SEARCH

by

Yeqin Zhang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Temperature Discovery Search (TDS) is a forward search method for computing or approximating the temperature of a combinatorial game. Temperature and mean are important concepts in combinatorial game theory, which can be used to develop efficient algorithms for playing well in a sum of subgames. A new algorithm TDS+ with five enhancements of TDS is developed, which greatly speeds up both exact and approximate versions of TDS. Means and temperatures can be computed faster, and fixed-time approximations which are important for practical play can be computed with higher accuracy than before.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Artificial Intelligence and Games

Games playing has been a research area since Artificial Intelligence (AI) was first introduced. Algorithms in AI originally were not good in dealing with ambiguous, ill defined real-world problems. Games, which are considered as an abstraction of real-world problems, have clear rules and goals and can also be incredibly complex. Thus many AI algorithms are tested in different kinds of games. The study of AI techniques can also lead to good strategies for game players and here are some famous game playing programs:

- The **backgammon** program *TD-gammon* developed by Tesauro [31] used temporal difference learning. It beat all other backgammon programs and its explored strategies had a great influence on the theory of backgammon playing.

- The **chess** program *Deep Blue* developed by IBM used $\alpha\beta$ search and a large endgame database. It defeated the world champion Garry Kasparov by 2 wins, 1 loss and 3 draws in 1997 [25].

- The **checkers** program *Chinook* developed by Schaeffer [24] is unbeatable and checkers is proved to be a draw [23].

- The **Go** program *Fuego* developed by Enzenberger et al. [8] uses Monte Carlo Tree Search (MTCS) and was the first program to beat a top professional in an even game on a 9×9 board.

Despite the global searches used in the computer programs for games above, another technique "Divide and Conquer" based on combinatorial game theory can be used for combinatorial games such as the game of **Amazons** and **Go**:

- The **Amazons** program *Arrow* developed by Müller [21] decomposed an Amazons board into a sum of independent subgames so it can search deeper in each local subgame. Using these techniques, Amazons on a 5×6 board was proven a first player win by Song and Müller [29].

1.2 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 covers the introduction of the game Amazons, background on combinatorial game theory, and different algorithms for sum games and algorithms for computing temperatures and means for a combinatorial game. This includes discussion of the Temperature Discovery Search (TDS) algorithm, on which our work is based. Chapter 3 explains how our new TDS+ method improves the original TDS algorithm. Chapter 4 evaluates the performance of TDS+ in several different test scenarios. Chapter 5 surveys other related work. Chapter 6 discusses some potential research topics on how to use TDS+ in different ways.

The main results of this thesis were accepted for publication in AAAI-15. An extended journal version is in preparation.

Chapter 2

Background

2.1 Game of the Amazons: An Example

The game of the Amazons (Amazons for short) is a well-studied combinatorial game. It was invented by Walter Zamkauskas of Argentina in 1988. It is a member of the territorial game family, a distant relative of Go and chess.

Amazons uses a rectangular board (usually 10 by 10 squares). The same number of black and white Amazons are placed on the board before a game starts. Usually there are four queens of each color, white moves first, and the players alternate moves thereafter until the game terminates when one of the players cannot make a move. The winner is the player who made the last move. Figure 2.1 shows the starting position of a 10×10 Amazons game.

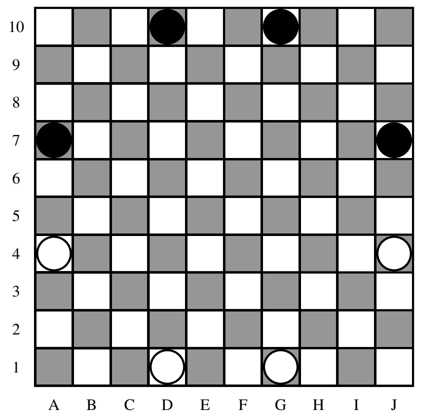


Figure 2.1: A 10×10 Amazons starting position.

Each move in Amazons consists of two parts:

1. **amazon move**. One player can choose one of his/her own amazons and move it exactly as a queen moves in chess, i.e., move one amazon from its origin square to another square in a straight line either orthogonally or diagonally, but it cannot cross or enter a square occupied by an amazon of either color or an *arrow*.

2. **arrow shot**. After one amazon is moved, it must shoot an *arrow* from its current square to another square, using another queenlike move.

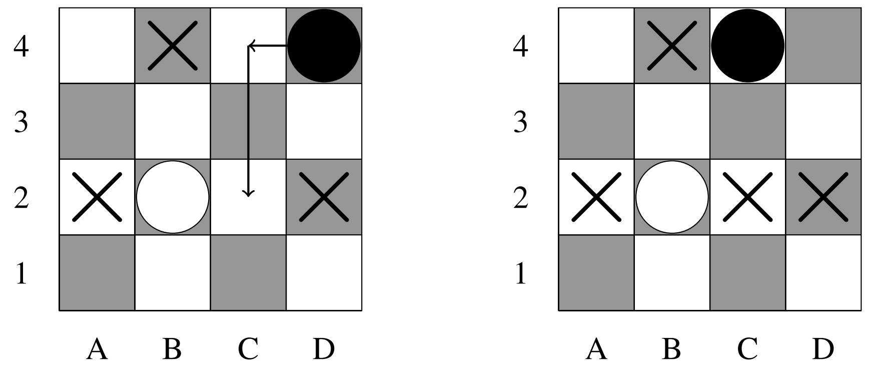


Figure 2.2: An example of how an Amazons move works.

Figure 2.2 shows an example move $D4 - C4 \times C2$.

Amazons is guaranteed to terminate since each move will fill an empty square and the total number of squares is finite.

Furtak et al. have shown that solving a generalized Amazons game is PSPACE-complete [9].

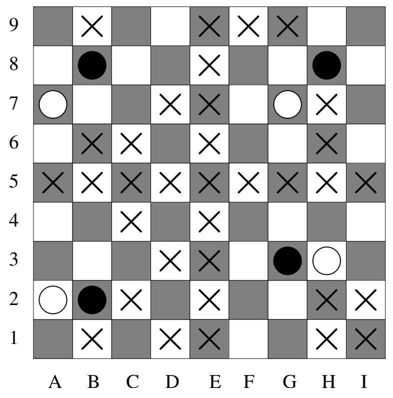


Figure 2.3: *Amazons* position with independent subgames in each corner, from [19].

In the end stage of an Amazons game, the board will often split into several parts such that each part is independent from the rest of the board. In Figure 2.3, from [19], solid walls of arrows separate a roughly 4×4 region in each corner to form four subgames. In terms of combinatorial game theory, the overall game position G can be represented as a *sum* of four simpler subgames $G_1 + G_2 + G_3 + G_4$.

2.2 Combinatorial Game Theory

Combinatorial game theory [5, 7] is the study of games that can be viewed as a *sum* of independent subgames. Each move changes exactly one subgame. The player able to make the last move overall wins. A recent textbook covering the theory is [26].

2.2.1 Combinatorial Games

Combinatorial Games are two-player games with no hidden information and no chance elements. They include children's games such as *tic-tac-toe* and *dots and boxes*; mathematical abstractions "played" on arbitrary graphs or grids or posets; and some of the deepest and best-known board games in the world, such as *Go* and *chess* [26].

Two players, often called **Left** and **Right**, play alternately and their moves affect the position in a manner defined by the rules of the game. The game terminates when one player cannot make a legal move. Under *normal* play the last player to move wins. In *misère* play the last player to move loses.

The game of Amazons is an example of a combinatorial game. According to the rules of Amazons, two players are playing on a board with perfect information and no random elements. The player who cannot make a move loses the game. All its properties satisfy the definition of a combinatorial game. Because of its decomposition property, the game of Amazons is a good testbed for studying combinatorial game theory.

Combinatorial game theory is most straightforward for *short games* or *loopfree games*. In the play of a short game, a position may never be repeated, and only a finite number of other positions can be reached.

A short game G is an ordered pair (G^L, G^R), where G^L and G^R are sets of "simpler" short games, the Left and Right options of G. It can be written as: $G = \{G^L | G^R\}$ [1].

2.2.2 Operations on Combinatorial Games

Combinatorial games can be summed and compared. The definition for summation, negation, subtraction and comparison $(<, =, >)$ can be defined by [1]:

$$
\begin{aligned}
G+H &\equiv \{G^L+H, G+H^L | G^R+H, G+H^R\} \\
-G &\equiv \{-G^R | -G^L\} \\
G-H &\equiv G+(-H) \\
G \geq H &\equiv \forall X \text{ Left wins } G+X \text{ whenever Left wins } H+X. \\
G \leq H &\equiv \forall X \text{ Right wins } G+X \text{ whenever Right wins } H+X. \\
G = H &\equiv G \geq H \text{ and } G \leq H. \\
G > H &\equiv G \geq H \text{ and } G \neq H. \\
G < H &\equiv G \leq H \text{ and } G \neq H. \\
G \mid\mid H &\equiv G \ngeq H \text{ and } G \nleq H.
\end{aligned}
$$

Let G and H be two combinatorial games with $G = \{G^L | G^R\}$, $H = \{H^L | H^R\}$.

$G+H$ means each player can either choose to play in G or H. $-G$ means players switched position/color, i.e. *Left* becomes *Right* and *Right* becomes *Left*. $G > H$ means *Left* has an advantage in G over H. $G||H$ means G is confused with H and neither game gives advantage for players. For example, game $G = \{0|0\}$, which is denoted as $*$, is confused with 0, thus $G||0$.

Every short game G has many equivalent forms. For example $*+* = 0$. But for all short games, there is a corresponding game called **canonical form** (or *simplest form*) of G. Every game G can be converted into one unique canonical form [1].

A combinatorial game in canonical form is said to be a **number** if each game in G^L is less than any game in G^R for $G = \{G^L | G^R\}$. A number is said to be an

integer if at least one of G^L and G^R is the empty set. From [1], we have:

$$0 \equiv \{|\}$$
$$n \equiv \{n-1|\}$$
$$-n \equiv \{|-(n-1)\}$$

Here a positive integer n means a game where player *Left* has n moves left while player *Right* has no moves available. Similarly for a negative integer $-n$, Right has n moves while Left has none.

In short games, numbers can only be dyadic numbers, i.e., the rationals whose denominators are the power of 2 [1].

Definition 1. *[1] Given $G = \{G^L|G^R\}$, $LS(G)$ (the left stop of G) and $RS(G)$ (the right stop of G) are defined in a mutually recursive fashion as:*

$$LS(G) = \begin{cases} x & \textit{if G is equal to a number x;} \\ \max(RS(G^L)), & \textit{otherwise;} \end{cases}$$

$$RS(G) = \begin{cases} x & \textit{if G is equal to a number x;} \\ \min(LS(G^L)), & \textit{otherwise;} \end{cases}$$

$LS(G)$ and $RS(G)$ can be viewed as the minimax value of G when G is played with Left/Right being the first player respectively, assuming the players stop playing as soon as the game becomes a number.

Definition 2. *[1] If G satisfies the condition that $-v < G < v$ for all positive numbers v, then G is called an infinitesimal.*

Therefore, the left and right stops of any infinitesimals are both 0. For example, $G = \{0|0\}$, is an infinitesimal.

2.2.3 Thermography

Definition 3. *[1] A game G cooled by t, G_t is defined as:*

$$G_t = \begin{cases} m & \textit{if } \exists t' < t : LS(G_{t'}) = RS(G_{t'}) = m; \\ \{G^L_t - t|G^R_t + t\}, & \textit{otherwise;} \end{cases}$$

Cooling G by t means that "you can move first in G if you donate t moves to your opponent".

Definition 4. *[26] The **Left** and **Right scores** of G, denoted by $L_t(G)$ and $R_t(G)$, are defined by*

$$L_t(G) = LS(G_t) \text{ and } R_t(G) = RS(G_t)$$

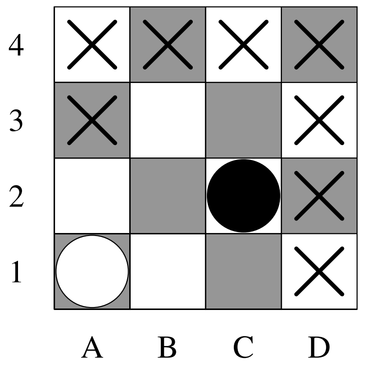
*regarded as functions of t. The **thermograph** of G is the ordered pair $(L_t(G), R_t(G))$.*

The functions $L_t(G)$ and $R_t(G)$ always intersect at some point (m,t) and are equal from all $t' > t$. We call this line from point (m,t) and above the mast of the thermograph and m,t are $m(G)$ (**mean** of G) and $t(G)$ (**temperature** of G) respectively.

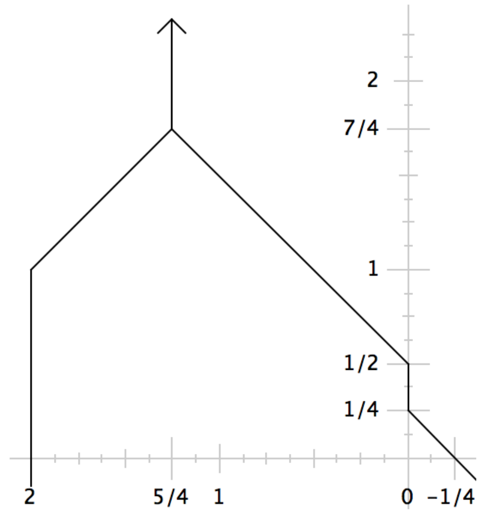
Temperature is the answer to the question: "How much is a move worth?" and mean is the answer to the question: "Who is ahead? And by how much?". Both notions are defined for a game without considering which player it is to move.

From a complete thermograph of G, $t(G)$ and $m(G)$ can be easily extracted as the coordinates of the base of the mast. This is one way to compute $t(G)$ and $m(G)$. In Figure 2.4b, $L_t(G)$ and $R_t(G)$ intersect at $(\frac{5}{4}, \frac{7}{4})$ which indicates that $m(G) = \frac{5}{4}$ and $t(G) = \frac{7}{4}$.

Besides $t(G)$ and $m(G)$, thermographs contain information about how G behaves at different temperatures, though not as much information as the canonical form. For example, infinitesimal difference between options are lost.



(a) An Amazons position G.



(b) Thermograph of G with mean $\frac{5}{4}$ and temperature $\frac{7}{4}$.

Figure 2.4: An Amazons position and its thermograph.

Figure 2.4 shows the thermograph of an Amazons position generated by CG-Suite (see website: `http://cgsuite.sourceforge.net`). The values on the horizontal and vertical axis represent mean and temperature values respectively.

It is hard to tell the mean or temperature of G from its canonical form:

$$G = \{4|2, \{3|0, \{\frac{1}{2}|0\}\}||\{0, *|-1\}, -\frac{1}{4}\}$$

while it is easy to tell from the thermograph. The mean value is just the mast value at the intersection, $\frac{5}{4}$ in the example and the temperature is just the temperature corresponding to the base of the mast, $\frac{7}{4}$ in the example.

2.2.4 Birthday of a Combinatorial Game

The birthday of a combinatorial game G, denoted as $b(G)$, is a measure of its recursive depth.

Definition 5. *[26] The* ***birthday*** *of a game (or value) $G = \{G^L|G^R\}$, denoted by $b(G)$, is 1 plus the maximum birthday of any game in $G^L \cup G^R$. In the base case, when $G^L = G^R = \emptyset$, $b(G) = 0$. G is said to be* ***born on day*** *n if $b(G) = n$, and it is* ***born by day*** *n if $b(G) \leq n$.*

0 is the only game born on day 0. Only four games, $1 = \{0|\}$, $-1 = \{|0\}$, $* = \{0|0\}$, and 0, are born by day 1. For short games born by day n, an upper bound of their temperature is $n-1$. For Amazons subgame G with m empty squares, $b(G) \leq m$. If $b(G)$ can be easily bounded from the set of possible temperatures, $t(G)$ will be constrainted. As we will see in Section 3.3.3, $t(G)$ lies in a much smaller set of possible temperatures for short games born by day n.

2.2.5 Enriched Environment and Coupon Stacks

It follows from [19] that an analysis for computing the temperature of a single subgame can utilize an *enriched environment* [3, 4] consisting of *elementary switches*, simple subgames of the form $v|-v$ for some number v. In $v|-v$, Black to move can gain v extra moves, while White to move can also gain v moves which is scored as $-v$. The mean of such a switch is $m(v|-v) = 0$ and its temperature is $t(v|-v) = v$ [5]. Sum games consisting only of such switches are easy - playing a switch with highest temperature v is always optimal. Following [3], coupon stacks can be extended to cover negative temperatures down to the lowest possible temperature of -1. A sum game consisting of a single, potentially complex, subgame G plus an enriched environment called a *coupon stack* can be used to determine both the mean and temperature of G [3].

In a sum game consisting of a single, potentially complex, subgame G plus an enriched environment, the trade-off now becomes much simpler to analyze: the choice is between playing in G and playing the switch with highest value v in the environment. If v is large, each player will take this profit rather than play in G. As the available values v become smaller, eventually a player will prefer to move in G instead.

Definition 6. *Given $\delta > 0$ and $t_{max} = n\delta$ for some integer n such that $n\delta \geq -1$, an extended coupon stack $C(t_{max}, \delta)$ contains coupons of value $t_{max}, t_{max} - \delta, \cdots, -1$, followed by a sufficiently large number k of final coupons of value -1, and a "balancing" coupon of value $-\frac{1}{2}$. If $t_{max} > -1$, the current player can take the top coupon of value t_{max} in $C(t_{max}, \delta)$. This changes the score of the game by $\Delta = t_{max}$ in that player's favor and leaves a shorter stack $C(t_{max} - \delta, \delta)$. When $t_{max} = -1$, a* *player can take a coupon of value* -1 *in this final coupon stack.*

The number k of extra coupons should be chosen large enough that there is always such a coupon available while play in G continues. A coupon stack $C = C(t_{max}, \delta)$ with $t_{max} \geq 0$ behaves like a combinatorial game of temperature $t(C) = t_{max}$ and mean $m(C) = 0$. For any $t_{max} \geq -1$, the *left score*, the minimax score with alternating play and Left going first, is $V(C, Left) = \lceil \frac{n}{2} \rceil \delta$ while the *right score* with Right going first is $V(C, Right) = -\lceil \frac{n}{2} \rceil \delta$.

[19] Let G be a finite loop-free combinatorial game and p be a player. In the view of thermography [5], a player should play in G at a temperature $t(G)$. However, in so-called *sente* situations a player has a threat that makes it possible to play G at a higher temperature. Let $\hat{t}(G, p)$ be the highest temperature such that player p can play in G without a loss in the thermographic sense. For loop-free games, this means p can play in G at any temperature in the range $[t(G) \ldots \hat{t}(G, p)]$.

2.3 Algorithms for Playing Sum Games

Algorithms for playing sum games emphasize local analysis of each subgame, since the computational complexity is much lower than when dealing with the whole sum at once. Three kinds of algorithms for playing sum games are global search algorithms, local search algorithms and combined global-local search algorithms.

2.3.1 Global Search Algorithms

The popular global search algorithms just consider sum games as a single game. Examples are $\alpha\beta$ search, which depends much on a good heuristic for position evaluation, and Monte Carlo tree search, which is widely used in computer Go.

$\alpha\beta$ **search** [16] is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. When applied to a standard minimax tree, it returns the same evaluation as minimax would, but prunes away branches that cannot possibly influence the final score. The move may change with use of transposition tables, etc. For branching factor b and search depth d, the best case time complexity of $\alpha\beta$ search is $\Theta(b^{\lceil d/2 \rceil})$. In games like Amazons, the branching factor can be very large, which results in a shallow search for a given time limit.

Monte Carlo Tree Search (MCTS) is a method for making decisions in artificial intelligence (AI) problems, typically move planning in combinatorial games. It combines the generality of random simulation with the precision of tree search [6]. Research interest in MCTS has risen sharply due to its spectacular success with computer *Go* [8, 10] and application to a number of other difficult games such as *Hex* [2], *Shogi* [22], and *Amazons* [17].

2.3.2 Local Search Algorithms

Local search algorithms can often search much deeper than global search does since they only search in one subgame, which has a much smaller branching factor. Local search usually cannot guarantee globally optimal move. Three local search algorithms discussed in [1] are:

Hotstrat: The player will always choose to play in the hottest subgame which is the subgame with the highest temperature.

Sentestrat: Define the *ambient temperature* as the lowest temperature of all previous positions; it starts out infinite. If the opponent has just moved in a component leaving it at a temperature above the ambient temperature, respond in that component; otherwise play hotstrat.

Thermostrat: Find the ambient temperature T and move in the component whose thermograph is widest at T. The difference between the optimal strategy and thermostrat is bounded by the largest temperature of the components. This strategy is used when the thermographs of each component can be easily calculated.

2.3.3 Global and Local Search Combination Algorithms

One global-local search was introduced in [20]. It is a hybrid algorithm that combines local temperature estimates with shallow global search.

[20] examines several different ways of combining local information with global search, including different ways to prune moves and order moves. The results show that a hybrid algorithm can perform much better than purely local search algorithms

like hotstrat or thermostrat. For example, in one scenario, given some temperature bound b, only the positions p with $t(p) > b$ are searched deeper, and positions p with $t(p) \leq b$ are evaluated statically.

2.4 Algorithms for Finding the Temperature of a Combinatorial Game

Both bottom-up and top-down algorithms can be used for finding the temperature of a combinatorial game.

2.4.1 Bottom-up Algorithms

The following works use bottom-up algorithms for finding the temperature of a combinatorial game:

- Snatzke [27] built a complete database of canonical form values and thermographs for Amazons positions within a set of small game boards by using bottom-up strategies which compute all results for smaller size Amazons games before computing larger size games.

- Tegos [30] also used bottom-up strategies to build a database of canonical form values and thermographs. Besides, he built another database called minimax database which stores the minimax value of every position for each player.

- Enzenberger's Amazons database module CGDB is part of the Arrow [21] code base. It is used for building Amazons playing and solving programs. It can compute the thermographs using bottom-up strategies. An extension by Song [28] can create databases for Amazons **blocker territories** which contain queens of only one color, and overlap with one or more active areas on some queens called blockers.

2.4.2 MTS: Mean and Temperature Search

Mean and Temperature Search (MTS) [12, 13] searches a subgame starting with alternating-first order and refines bounds on means and temperatures up to convergence. The main limitation of MTS is the requirement that all game positions of temperature zero and below can be statically recognized and evaluated. In many combinatorial games, including Amazons, this is not feasible.

From [12, 13], the outline of MTS is similar to an $\alpha\beta$ search. $M^L(G)$, $M^U(G)$, $T^L(G)$, and $T^U(G)$ denote the lower and upper bound of the mean and temperature of a position G. Before the search starts, these values are initialized for all positions: $M^L = -\infty$, $M^U = \infty$, $T^L = 0$, $T^U = \infty$. During each run of the search, a new terminal node will be visited and the values of all nodes on the path from the root to the terminal node will be updated. The search process continues until $M^L = M^U$ and $T^L = T^U$ at the root. The search may terminate while there are still unvisited nodes, which corresponds to pruning in $\alpha\beta$ search. In the worst case, as in $\alpha\beta$ search all terminal nodes will be visited.

2.4.3 TDS: Temperature Discovery Search

Temperature Discovery Search (TDS) [19] can handle any short (loopfree) game including those with positions of nonpositive temperature.

Both MTS and TDS are forward search algorithms. They do not need to construct the complete thermographs of all follow-up positions by bottom up analysis.

TDS is a forward search algorithm based on $\alpha\beta$ search of the sum $G+C$, where G is the game to be analyzed and C is a coupon stack. Taking a coupon of value v is represented by $C(v)$. With a small-enough δ and large-enough t_{max}, TDS computes exact means and temperatures for loopfree combinatorial games. TDS can *fail* if the temperature t_{max} of the largest coupon is too small. This is indicated by a principal variation (PV) of the $\alpha\beta$ search which starts with a move in G, not a coupon.

Using an Enriched Environment to Determine $t(G)$ and $m(G)$ [19]:

First the maximum possible temperature, t_{max} which $t(G)$ could be, is determined. Then a search starts in the game $G+C(t_{max},\delta)$ for a player p to compute

$V(G+C(t_{max},\delta),p)$.

1. *Determine the mean*: Assume that p is the player to move. According to a theorem in [3], if the value δ in C is sufficiently small, then $V(G+C,p) = m(G) + V(C,p)$. Since $V(C,p)$ is known (see Section 2.2.5), $m(G)$ can be determined.

2. *Determine the temperature*: Observe the move sequence in the principal variation (PV) returned by the search. If $t(C) > \hat{t}(G,p)$, then the first moves in the PV will all be in C. At some time when $t(G) \leq t(C) \leq \hat{t}(G,p)$, the first move in G will appear in the PV. This can be used as a starting point of an iterated search process to determine $t(G)$. $t(G)$ is the lowest temperature at which optimal play can switch from C to G. For example, if the first move in G was played between the coupons of value t and $t-\delta$, The first estimate of $t(G)$ is set to t, since $t(G) \leq t$. If the minimax score stays the same even if the next lower coupon of value $t-\delta$ is played before the first move in G, then the temperature estimate can be lowered by δ. To check this, search is modified to play all the coupons down to $C(t-\delta)$ before a move in G is allowed. If the minimax score remains the same, then taking $C(t-\delta)$ first was not a loss, compared to the previous search where G was played before $C(t-\delta)$. Therefore the estimate can be decreased to $t := t-\delta$, and the process is repeated. Otherwise, if the minimax score changes, then taking $C(t-\delta)$ was nonoptimal, and the temperature is $t(G) = t$.

The process above is called temperature discovery search (TDS) and Algorithm 1 shows the pseudo code.

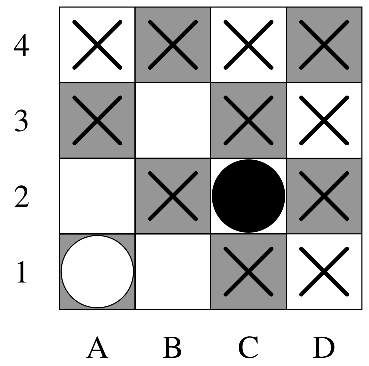


Figure 2.5: Amazons example $G = \{2 | -\frac{1}{2}, \pm 1\}$.

Figure 2.5 [19] shows a local Amazons position G with Black = Left to move first. The search of $G+C$ with an extended coupon stack $C = C(t_{max} = 17, \delta = \frac{1}{8})$ yields the minimax score $V(G+C, Left) = \frac{15}{8}$. The principal variation (best line of play-PV) for Figure 2.5 is:

1. $C(\frac{17}{8})$ **2.** $C(\frac{16}{8})$ **3.** $C(\frac{15}{8})$ **4.** $C(\frac{14}{8})$ **5.** $C(\frac{13}{8})$
6. $C(\frac{12}{8})$ **7.** $C(\frac{11}{8})$ **8. A1–A2×B1 9.** $C(\frac{10}{8})$ **10.** $C(\frac{9}{8})$
11. $C(\frac{8}{8})$ **12.** $C(\frac{7}{8})$ **13.** $C(\frac{6}{8})$ **14.** $C(\frac{5}{8})$ **15.** $C(\frac{4}{8})$
16. $C(\frac{3}{8})$ **17.** $C(\frac{2}{8})$ **18.** $C(\frac{1}{8})$ **19.** $C(0)$ **20.** $C(-\frac{1}{8})$
21. $C(-\frac{2}{8})$ **22.** $C(-\frac{3}{8})$ **23. C2–B3×C2**.

Algorithm 1 TDS

```
1:  function TDS($G$)
2:      $t$ := MaxPossibleTemperature($G$)
3:      $\delta$ := SmallEnoughDelta($G$)
4:      (v, sequence) := $\alpha\beta(G+C, p, t)$        ▷ solve $G+C$ for player $p$, no board
    move when top coupon is greater than t
5:      $T$ := GetTemperatureFromSequence(sequence)
6:      $M := v - v(C, p)$
7:      while True do
8:          (newv, newsequence) := $\alpha\beta(G+C, p, T-\delta)$
9:          if newv $\neq$ v then
10:             $M := v - v(C, p)$
11:             break
12:         else
13:             v := newv
14:             sequence := newsequence
15:             $T$ := GetTemperatureFromSequence(sequence)
16:         end if
17:     end while
18:     return $(T, M)$
19: end function
```

Approximate TDS uses a larger value of δ than the required small enough δ. It was shown to yield excellent approximations of means and temperatures even with relatively large δ such as $\frac{1}{2}$. In experiments on sums of Amazons positions, a heuristic TDS-based algorithm outperformed global $\alpha\beta$ search for sums of Amazons subgames.

Let G be a (sub)game to be analyzed. A *search state* in TDS is defined as $S(g, c, \Delta, toPlay)$, where g is the current game position reached by play from G, c is the current coupon stack, Δ is the aggregate score of all coupons already taken, with coupons taken by White counted negative, and *toPlay* is the color to play next. A move in g changes the state to a board position g', while a move in c changes both the coupon stack and Δ. Any move also changes *toPlay* to the opponent.

Chapter 3

TDS+: Improving Temperature Discovery Search

3.1 A Motivating Example

The main problem of applying the original TDS algorithm in practice is its time complexity when scaling up - either to larger subgames, or to higher precision using smaller δ. As an illustration, consider the game G in Figure 2.5 with temperature $t(G) = \frac{5}{4}$ where TDS scales badly for small δ. As an approximation algorithm with a relatively large $\delta = 1/2$ and setting $t_{max} = 2 + \delta$, TDS is reasonably fast and computes the following principal variation (PV) with Black going first:

1. $C(\frac{5}{2})$ **2.** $C(\frac{4}{2})$ **3.** $C(\frac{3}{2})$ **4. A1–A2**$\times$**B1 5.** $C(\frac{2}{2})$
6. $C(\frac{1}{2})$ **7.** $C(0)$ **8.** $C(-\frac{1}{2})$ **9. C2–B3**$\times$**C2**.

The first move on the board is played at move 4, between coupons of value $\frac{3}{2}$ and 1. This represents a good approximation to $t(G) = \frac{5}{4}$ and is achieved with a relatively fast 9-ply search. However, computing the exact temperature requires setting $\delta = 1/8$. This leads to a deep 23-ply search as shown in the previous chapter.

The optimal line of play contains two long coupon-taking subsequences but only two moves (8 and 23) on the game board. Moves 1–7 are all coupons since the initial temperature $t_{max} = \frac{17}{8}$ of the coupon stack is considerably higher than the board temperature of $\frac{5}{4}$. After the first board move **8. A1–A2**$\times$**B1**, the temperature of the board drops to $-\frac{1}{2}$, and therefore all moves 9–22 are again coupons.

In the simple standard model with fixed branching factor b and fixed depth d, the best case time complexity of $\alpha\beta$ search is $\Theta(b^{\lceil d/2 \rceil})$. Compared to an $\alpha\beta$ search of G without a coupon stack, b increases by one in TDS for the added coupon move. However, d increases by the number of coupons that need to be taken before reaching a terminal position in G, which can be very large when δ is small and t_{max} increases. The number of coupons in a coupon stack is about $(t_{max}+1)/\delta$, plus possibly several final coupons of value -1. Since the branching factor b is determined by the game, the current work focuses on how to reduce the search depth d.

One important improvement implemented in the original TDS, and used in the example above, is that as soon as a recognized terminal position is reached in G, TDS stops the search and computes the alternating-play value of the remaining coupons. To improve the speed of TDS in practice, reducing the search depth is essential. The TDS+ algorithm developed in this thesis achieves this *while retaining correctness*. The improvements to TDS developed in this thesis are based on three main insights:

1. In the PV of TDS, long consecutive coupon move sequences exist. So reducing the effective search depth can be achieved by avoiding long sequences of coupons, both at the beginning and in the middle of a search.

2. Fast pre-searches with a large value of δ can be used to quickly gain information about a game, in order to set up the final, expensive search as well as possible.

3. The transposition table is not fully used in TDS. The fact that a sum $G+C$ is searched can be used for strong algorithm-specific improvements to the transposition table, which allow much better re-use of information compared to the "plain $\alpha\beta$" transposition table used in the original TDS algorithm. However, some care is needed to handle this re-use correctly, as discussed in the next section.

3.2 Re-using State Information and Solving a TDS-specific Graph History Interaction Problem

The graph history interaction (GHI) problem occurs when the outcome of a game depends on the path (history) of moves from the initial state. The most frequent example of this problem is position repetition. Surprisingly, GHI can appear when searching $G+C$ even when a game G itself has no history dependency.

Play of $G+C$ ends either in a *normal terminal position*, where the value of G can be statically recognized, or in a *pseudo-terminal position* (PTP), where both players took a -1 coupon as their last move, indicating their unwillingness to continue play. PTP are evaluated as 0 by the simplicity rule of combinatorial game theory [5, 19]. PTP can cause a GHI problem as follows:

Let δ be fixed, and let $c_{-1} = C(-1,\delta)$ be a *final* coupon stack with $t_{max} = -1$. Consider playing $G+c_{-1}$ with Black to play, when there exist moves a for Black and b for White such that the resulting board position G' is the same after either sequence ab or ba played from G. Then the move sequence ending with successive -1 coupons:

1. Black a **2.** White b **3.** Black $C(-1)$ **4.** White $C(-1)$

is a PTP which is evaluated as 0, while the sequence:

1. Black $C(-1)$ **2.** White b **3.** Black a **4.** White $C(-1)$

is not and might have a different minimax score. Both sequences result in identical board positions G' and coupon stacks c_{-1}, but their evaluation is not the same in general. In the original TDS algorithm, this problem is avoided since the transposition table is only used in a very conservative way: states reached after consecutive -1 coupons are never stored or looked up in a table. The more aggressive use of tables in TDS+ requires handling this GHI problem. While efficient general solutions exist [14], for the current special case it suffices to slightly extend search states by storing the number of consecutive -1 coupons taken as the most recent moves. Instead of storing both states resulting from the two sequences above as $S(G',c_{-1},\Delta,Black)$, which would cause a GHI problem, the extended states become distinct: $S(G',c_{-1},\Delta,Black,2)$ and $S(G',c_{-1},\Delta,Black,1)$.

3.3 Five Enhancements to TDS

3.3.1 Conditional Move Generation

Algorithm 2 Conditional Move Generation using a temperature-dependent Skip() test.

```
function CONDITIONALGENERATE($G$, $C$, Skip)
    $t$ := MaxTemperature($C$)
    if Skip(t, $G$) then
        return $\{C(t)\}$
    else
        return $\{C(t)\} \cup$ Generate($G$)
    end if
end function
```

Two of the enhancements below work by suppressing move generation in the game G for specific temperatures t. In principle, coupons at these temperatures could be removed from the coupon stack, but the bookkeeping for stacks with nonuniform temperature differences becomes messy. In Algorithm 2, a uniform stack is retained, but move generation in G is skipped at these temperatures, resulting in a very fast unbranched search step. Enhancements E_2 and E_4 below utilize this approach, with different **Skip** functions.

The following five enhancements lead from TDS to an improved algorithm TDS+.

3.3.2 E_1: Fast Pre-searches with Decreasing Values of δ

The original TDS sets $t_{max} = bound + \delta$, where *bound* is a game-specific bound on the maximum possible temperature. In Amazons, a safe bound for a position with n empty squares is $n-1$ (see Section 2.2.4). However, the temperature of most positions is much lower. As in Figure 2.5, searching with a larger δ is much faster and can be used to obtain a better t_{max} estimate. Extensive empirical testing showed that the estimated temperature returned from such searches never underestimates by much, giving rise to the 2δ**-Conjecture**.

Conjecture 1. *Let $t_\delta = TDS(G, \delta, t_{max})$ be the approximate temperature computed for some δ. Then the true temperature $t(G)$ is upper bounded by $t(G) \le t_\delta + 2\delta$.*

Algorithm 3 shows TDS with enhancement E_1. G is searched repeatedly with decreasing values of $\delta = 1, \frac{1}{2}, \cdots, \frac{1}{2^n}$, while adapting t_{max} along the way. Since the 2δ-conjecture is unproven, the call to TDS in Line 7 of the algorithm could possibly fail. In this case, the algorithm needs to re-search with larger t_{max} until it succeeds. However, this case has never happened in thousands of experiments. Choosing a lower position-dependent t_{max} means fewer coupons in the final, most expensive search. In the ideal case the PV starts with a single coupon, followed by a move in G.

Algorithm 3 TDS_1: Pre-searches with δ from 1 to 2^{-n}

```
 1: function TDS_1($G, n$)
 2:     $\delta := 1$
 3:     $t_{max} :=$ safe_bound($G$)                          ▷ $n-1$ in Amazons
 4:     while $\delta > 2^{-n}$ do
 5:         isFail := true
 6:         while isFail do                                ▷ Failure handling
 7:             $t_\delta :=$ TDS($G, \delta, t_{max}$, isFail)
 8:             if isFail then
 9:                 $t_{max} := t_{max} + \delta$
10:             end if
11:         end while
12:         $t_{max} := t_\delta + 2\delta$                      ▷ Adapt $t_{max}$ for next iteration
13:         $\delta := \delta/2$
14:     end while
15:     return TDS($G, \delta, t_{max}$)
16: end function
```

3.3.3 E_2: Avoid Search at Impossible Temperatures

For fixed n, the set of games born by day n is finite. Theorem 1 in Appendix A characterizes a set $T_n = \{-\frac{1}{2^b}, 0, \frac{1}{2^b}, \frac{3}{2^b}, \ldots, a + \frac{1}{2^b} | 0 \leq a \leq n-2, 0 \leq b \leq n-1\}$, which contains all temperatures of games born by day n. The following function can be used to check whether a temperature is possible or not.

Algorithm 4 Skipping Impossible Temperatures

```
1: function SKIP-IMPOSSIBLE-T($t, G$)
2:     $n :=$ BirthdayUpperBound($G$)              ▷ $n$ empty squares in Amazons
3:     return $t \notin T_n$
4: end function
```

Enhancement 2 directly applies this theorem using the SKIP-IMPOSSIBLE-T function in Algorithm 4 as the argument *Skip* in CONDITIONALGENERATE of Algorithm 2. This test requires an estimate of the birthday of a game G. Especially for small δ, many temperatures can be skipped.

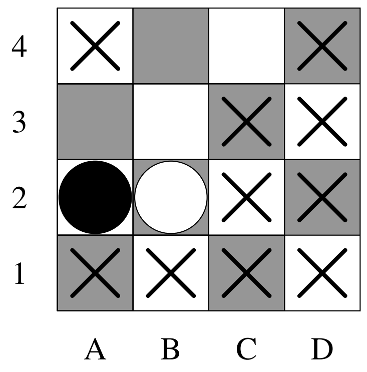


Figure 3.1: Amazons example $G = \pm(1, \{2|\frac{1}{2}, \{1|*\}\})$, where E_2 can help skip many coupons.

Figure 3.1 shows an example of how E_2 performs. First, TDS_1 lowers t_{max} to $\frac{22}{16}$ (details not shown). Then, the PV of the final TDS search is as follows:

1. $C(\frac{22}{16})$ **2.** $C(\frac{21}{16})$ **3.** $C(\frac{20}{16})$ **4.** $C(\frac{19}{16})$ **5. B2–B3×B2**
6. $C(\frac{18}{16})$ **7.** $C(\frac{17}{16})$ **8.** $C(\frac{16}{16})$ **9.** $C(\frac{15}{16})$ **10.** $C(\frac{14}{16})$
11. $C(\frac{13}{16})$ **12.** $C(\frac{12}{16})$ **13. B3–A3×B3**

For 7 out of the 11 coupons, search is skipped by E_2 along the PV. These coupons are $C(\frac{21}{16})$, $C(\frac{19}{16})$, $C(\frac{18}{16})$, $C(\frac{17}{16})$, $C(\frac{15}{16})$, $C(\frac{14}{16})$, and $C(\frac{13}{16})$. Note that in other branches of the search, the set of skipped coupons may be different, since it depends on the birthday upper bound of the current board position.

3.3.4 E_3: Generalized Transposition Table

The original TDS implementation uses a standard hash table to recognize transpositions in its $\alpha\beta$ search. The design of its hash function for coupon stacks did not allow re-use of information between searches. The generalized transposition table of TDS+ improves upon TDS in three ways: First, TDS+ computes the hash code for a stack c by first defining a hash function mapping each temperature t to a hash code $h(t)$, then defines the hash code of c as the bitwise xor of the codes of all coupons with temperature $t > -1$. Second, in order to deal with GHI, the hash code encodes the number of consecutive -1 coupons taken as the most recent moves. This allows re-use of hash table entries between different searches.

Third, TDS+ generalizes the entries in the hash table as follows: The minimax value of a full state $S(g, c, \Delta, \text{toPlay}, \text{nuFinal-1Coupons})$ is the sum of the aggregate value Δ of coupons taken so far, and its remaining value, which depends on the other state variables g, c, *toPlay* and nuFinal-1Coupons. In the TDS+ hashtable, states are stored without encoding Δ. The value of Δ is kept up to date incrementally in a search while traversing the game tree, and is added to each value retrieved from the hash table. In this way, a state s' that has a different past history in terms of coupons taken but is the same otherwise as a state s can be used to compute the value of s without search.

3.3.5 E_4: Recursive TDS

While enhancement E_1 is designed to lower t_{max} and avoid search at too-high temperatures at the beginning of the search, the same idea can be applied recursively after each move on the board, since the temperature may have dropped significantly, as in Figures 2.5 and 3.1. Enhancement E_4, shown in Algorithm 5, recursively calls TDS_1 to compute a t_{max} estimate at every position during the search. In case of a temperature drop, this approach can skip many coupons in the top-level search.

When combining the SKIP-RECURSIVE function with the SKIP-IMPOSSIBLE-T of enhancement E_2, the algorithm first uses the SKIP-IMPOSSIBLE-T to check if the temperature is possible or not. It also records the maximum possible temperature for the current board position and saves it in a separate board hash table, which stores useful information for board positions independent of the state of the coupon stack, also including the score of each color by the static evaluation function. Each time the board hash table is checked first to see if this position was searched before. If so, the maximum possible temperature stored is used directly without searching this position again. Then SKIP-RECURSIVE is used to check the temperature again.

Algorithm 5 Lower the temperature at internal nodes

```
1: function SKIP-RECURSIVE($t, G$)
2:     $n := BirthdayUpperBound(G)$
3:     return $t >$ TDS$_1(G, n)$
4: end function
```

3.3.6 E_5: Improved Handling of PTP States

With E_5, PTP states (see Section 3.2) reached after two consecutive -1 coupons are recognized as solved positions. Without E_5, the same $G+C$ reached after two consecutive -1 coupons may be visited many times during the search and the algorithm needs to check whether it is a solved position every time. With E_5, the information of whether the state is reached by two consecutive -1 coupons is also stored in the transposition table. The algorithm can use this information to avoid duplicate checking, which leads to a speedup.

3.3.7 The TDS+ Algorithm

The TDS+ algorithm uses all enhancements $E_1, E_2, \ldots, E_5$. It simply calls the function TDS from Algorithm 1 with the following changes:

- Move generation in $\alpha\beta$ search uses Algorithm 2 to avoid board move generation at impossible temperatures which is determined by enhancements E_2 and E_4. E_1 is used in E_4 as shown in Algorithm 5.

- By using enhancement E_3, the transposition table now stores better information than the old algorithm. It can also use E_5 to deal with the GHI problem.

3.3.8 Using TDS+ When Bounds for Birthday Are Unknown

In E_1, E_2, and E_4, we assume that a bound t_{max} for temperature and a bound $b(G)$ for birthday exist and t_{max} is initially set to this bound according to the birthday. For the game of Amazons, the birthday for each position can be easily bounded according to the number of empty squares, but for other combinatorial games, the birthday cannot be bounded easily.

Here are some suggestions for use when bounds for birthday are unknown.

- When t_{max} cannot be easily bounded: first run TDS/TDS+ using a guess for t_{max}. If $t_{max} > t(G)$, the algorithm will work correctly; if $t_{max} \leq t(G)$, and the first move in the PV is a board move, then run TDS/TDS+ again using a larger t_{max} by adding 1 or some other good step to the previous t_{max} and keep following this process until the search returns a PV starting with a coupon.

- When $b(G)$ cannot be easily bounded: the use of E_2 is restricted: from Theorem 1, negative temperatures can only be of the form $-\frac{1}{2^n}$.

- When small enough δ cannot be determined: in this case, we can only generate an approximation for temperature and mean by successively setting δ smaller and smaller. This approximate version can also use an approximation for birthday $b(G)$ and therefore use E_2 again.

Chapter 4

Experiments and Results

All experiments use the game of Amazons and are performed on a 2.4 GHz Intel Xeon. The maximum memory in the experiment is 80 MB for the hash tables of entries.

4.1 Improvement from Individual Enhancement

TDS+ corresponds to the original TDS algorithm plus all enhancements E_1–E_5. This section investigates the performance of different subsets of enhancements. The presence of enhancement E_i is indicated by adding i to the subscript of TDS. For example, TDS_{13} uses E_1 and E_3, and TDS+ is TDS_{12345}.

Not all subsets are meaningful, since E_4 requires both E_1 and E_3, while E_5 requires E_3. E_4 includes E_1, and E_4 requires E_3 since E_4 is used recursively which leads to many calls of the same searches for the same position. Using the hash table (E_3) can avoid duplicate searches.

The test set contains 600 cases from a complete database of 4×4 Amazons positions with one queen each: 17 cases with two empty squares, 33 cases with 14 empty squares, and 50 test cases each for 3 to 13 empty squares. They were randomly sampled from the database. (See Appendix B for details.) Experiments were performed to test interesting subsets of enhancements, including each enhancement in isolation, as well as a *leave-one-out* setting. Figures 4.1 and 4.3 show the results in terms of *coverage*, or number of problems solved, with different time limits. Figure 4.3 shows the comparison of enhancements in a *leave-one-out* setting. Each version of TDS in Figure 4.3 except TDS+ lacks one single enhancement.

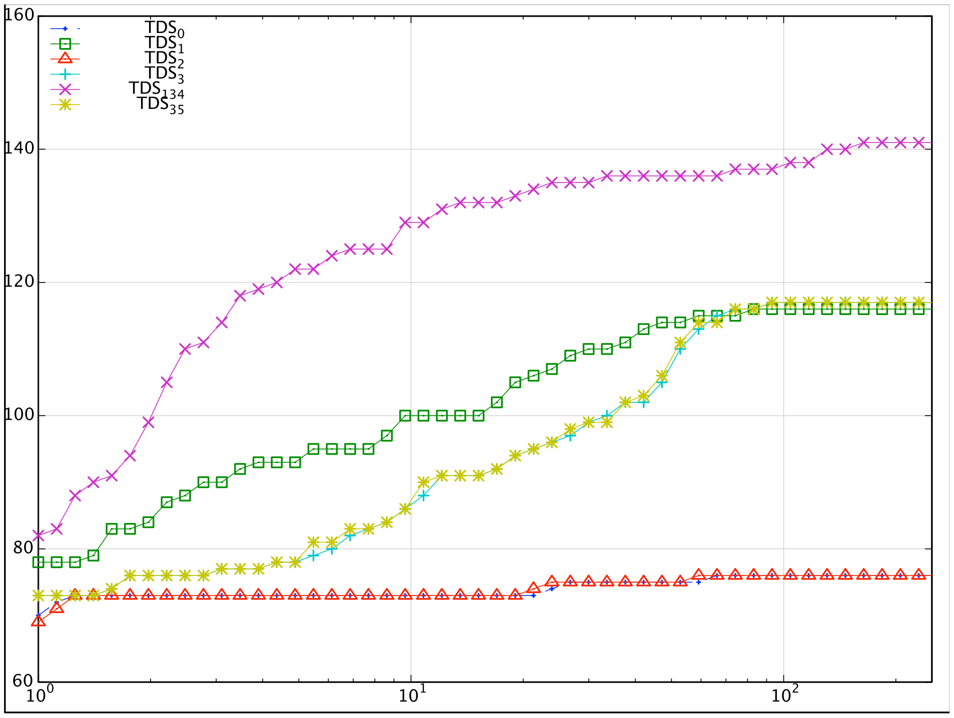


Figure 4.1: Comparison of each enhancement in isolation.

The x-axis in Figures 4.1, 4.2, 4.3 and 4.7, which is on a log scale, represents the time limit (in seconds) for each test case, and the y-axis represents the number of solved test cases in for a given time limit.

Discussing the contribution of each enhancement from the results in the figure, TDS$_1$ already solves many more test cases than TDS. However, its scaling with higher time limits is also poor. Comparing TDS$_{235}$ with TDS$_{1235}$ shows that E_1 is also very strong in combination.

Results for TDS$_2$ show that E_2 alone helps little. However, combined with other enhancements it works very well for more complex test cases, as shown by the big difference between TDS$_{1345}$ and TDS$_{12345}$ for longer time limits.

The individual strength of E_3 is similar to E_1, as seen when comparing TDS$_1$ and TDS$_3$, and also TDS$_{235}$ and TDS$_{12}$. These two combine very well in TDS$_{13}$, and also in TDS$_{1235}$ compared to both TDS$_{235}$ and TDS$_{12}$.

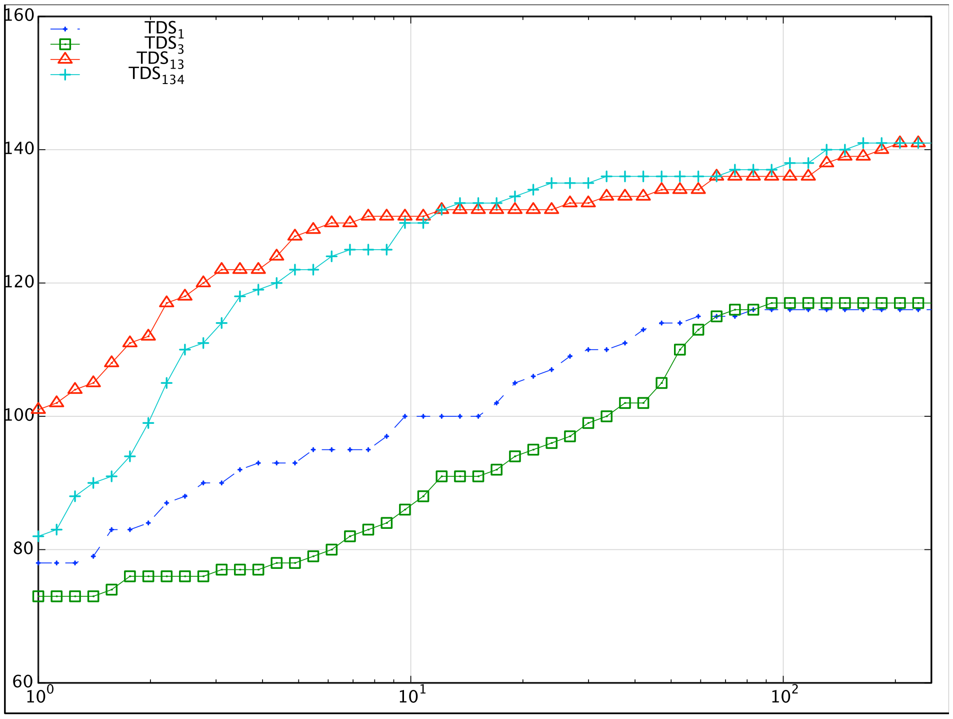


Figure 4.2: Combining enhancements E_1, E_3 and E_4.

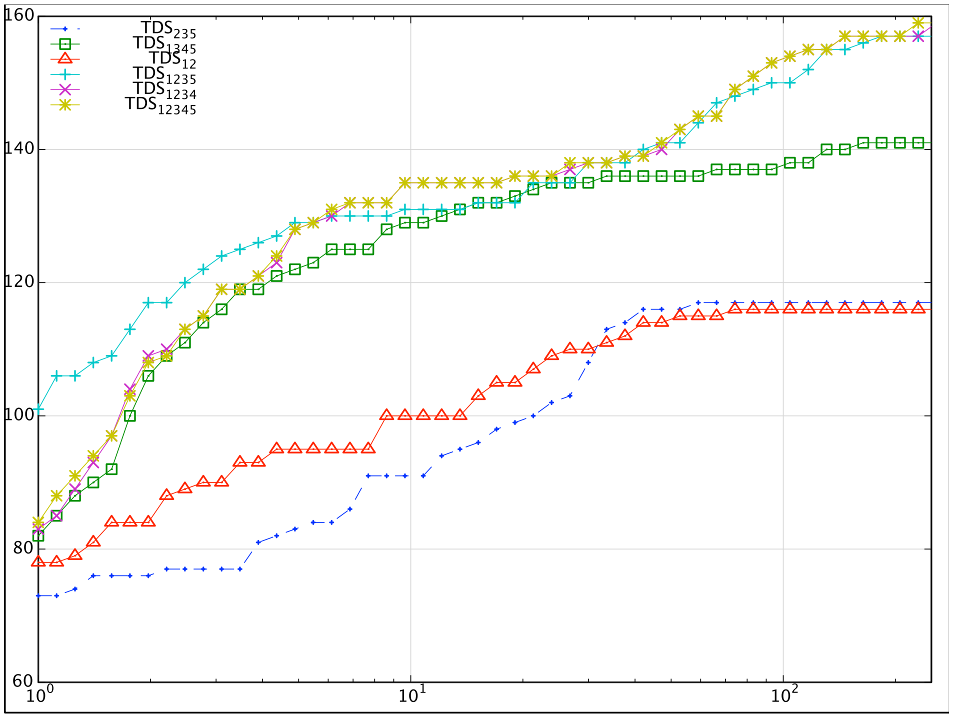


Figure 4.3: Comparison of enhancements in leave-one-out setting.

Adding E_4 is a strong improvement over E_3 alone but when adding E_4 to TDS_{13}, the improvement is not so much as seen in Figure 4.2.

TDS_{12345} and TDS_{1235} also have similar coverage. TDS_{12345} can solve more of the high temperature test cases.

Figure 4.4 shows details. For high temperature test cases with $t(G) \geq 2$, TDS_{12345} is always faster than TDS_{1235}. Since larger size test cases (6 or more empty) are not all complex test cases, TDS_{1235} can finish some of them faster than TDS_{12345} does.

The improvement from E_5 is modest, but visible in a few data points in Figure 4.3. To give a quantitative indication, TDS_{12345} expands 0.5% fewer nodes than TDS_{1234} over the set of all 50 test cases with 5 empty squares.

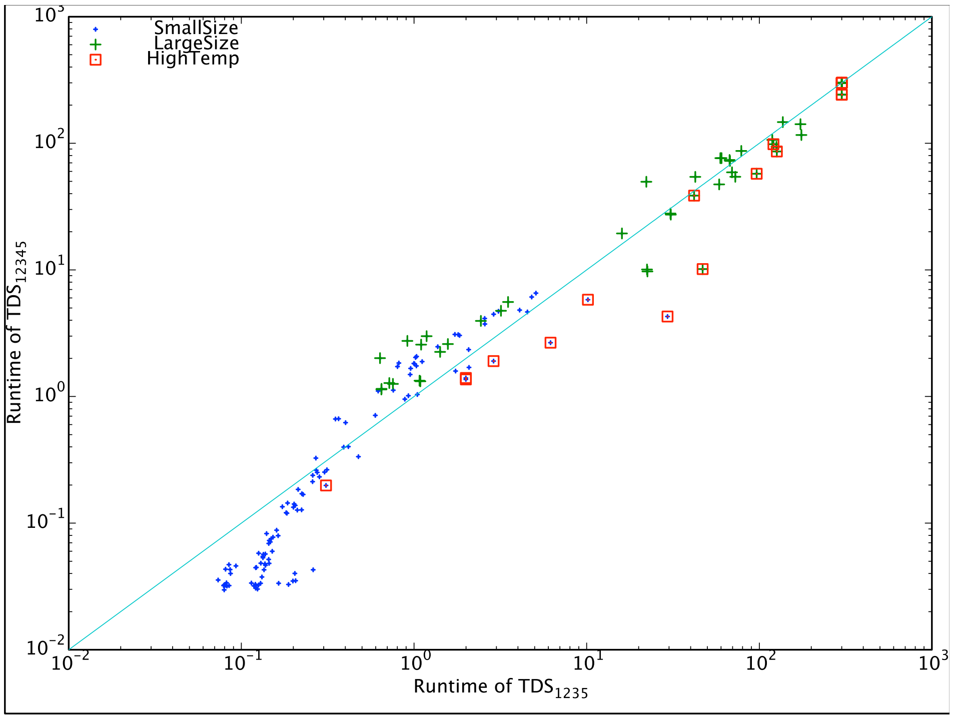


Figure 4.4: Runtime comparison, TDS_{1235} vs TDS_{12345}.

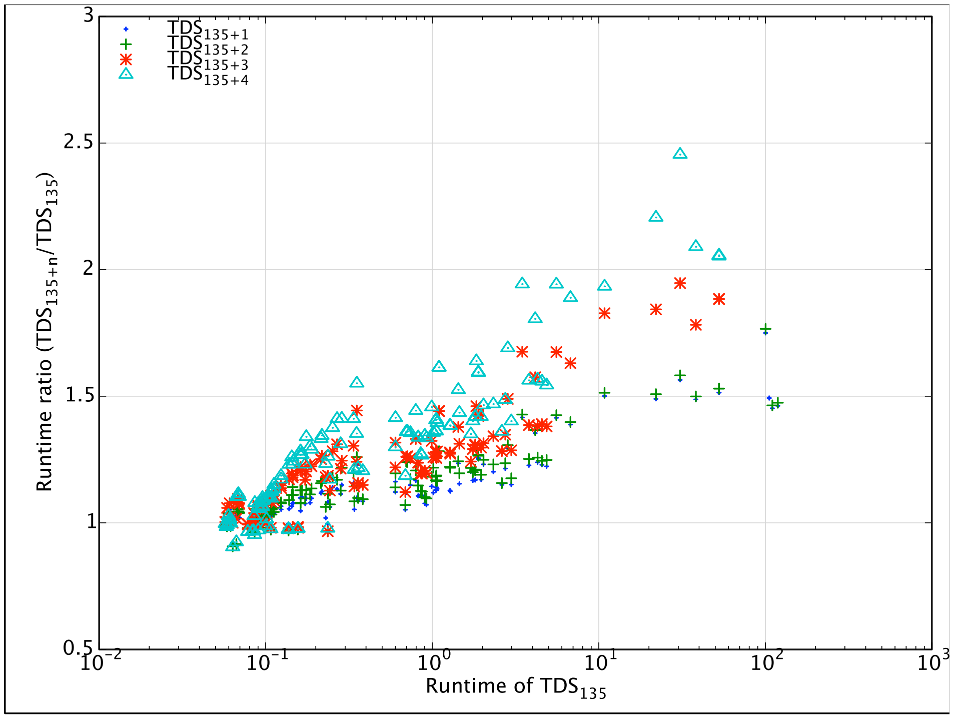


Figure 4.5: Runtime comparison, TDS_{135} vs. TDS_{135+n}.

Figure 4.5 gives further evidence that reducing the number of top coupons as in E_1 is important. In this experiment, TDS_{135+n} is TDS_{135} but with n extra top coupons of value $t_{max}+\delta, \cdots, t_{max}+n\delta$ added after establishing t_{max} using TDS_1. For all test cases with runtime over 1 second, TDS_{135} is fastest.

Only cases where all experiments finished within the time limit are shown. Out of the 144 test cases that completed with TDS_{135}, the number of extra timeout cases was 3, 5, 8, 8 for $n = 1\ldots4$. Runtimes are plotted on a log scale in Figures 4.4 and 4.5.

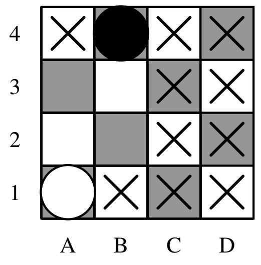


Figure 4.6: Amazons example where adding E_4 is much better. $t(G) = 3$, $m(G) = 0$.

Figure 4.6 shows an example where TDS_{12345} performs much better than TDS_{1235}.

TDS_{1235} expands 800,751 nodes while TDS_{12345} only expands 31,041 nodes and is about 7 times faster. It cost 29.39s for TDS_{1235} while it only cost 4.28s for TDS_{12345}. The reason is that after the first move on the board, the temperature drops from 3 to -1. E_1 can only lower t_{max} for the starting position while E_4 can lower t_{max} for any intermediate board position.

4.2 2δ-Conjecture Experiment

The 2δ-Conjecture is used for enhancement E_1, see Section 3.3.2.

This experiment tests how much can be gained if only one δ is added to t_{max} instead of using the 2δ-Conjecture on TDS_{1235}. The 2δ-Conjecture is used in all other experiments since it was found empirically that the 2δ-conjecture never failed.

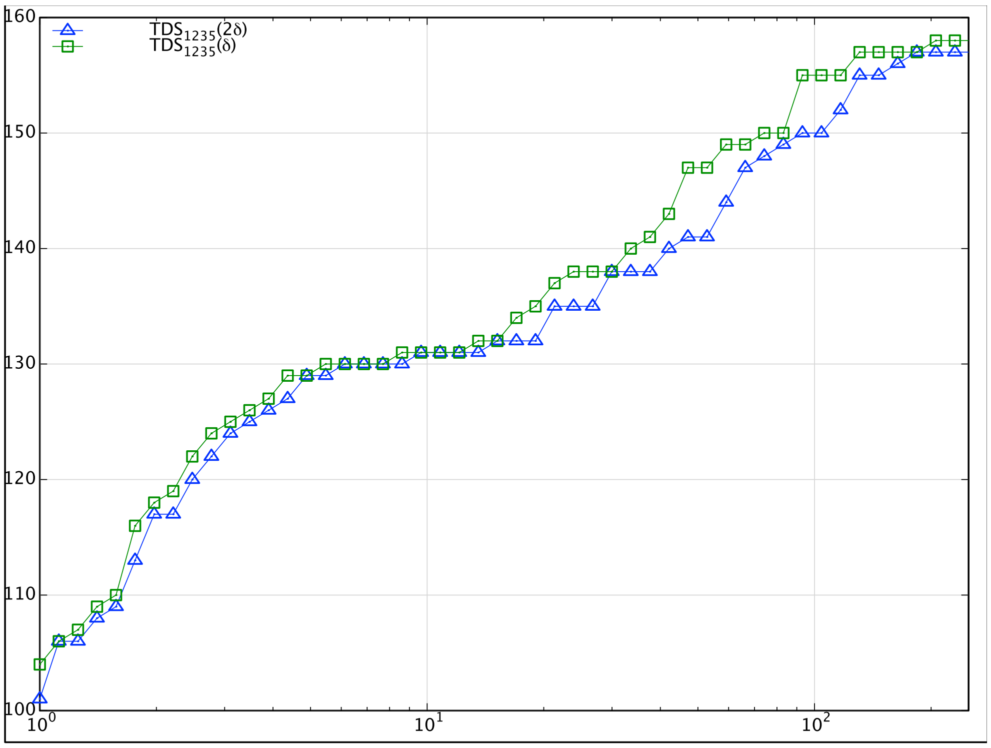


Figure 4.7: TDS_{1235} when using $\delta/2\delta$ conjecture.

In Figure 4.7, the results show that TDS_{1235} can be improved a little when adding only one δ to t_{max}in Line 12 of Algorithm 3, which means 2δ needs to be added to t_{max} only very few times.

The one δ version of TDS$_{1235}$ is used for all searches beginning with the initial large δ to the final, small enough δ.

It is a trade off to add one δ or 2δ to t_{max} since adding one δ can shorten the search depth while adding two δ can avoid the re-search assuming the 2δ-Conjecture never fails.

In the experiment, for all size 4 to 6 (2 to 4 empty squares) test cases, only two cases have a PV with exactly 1 coupon before the first board move in the final search. For all other 131 passed test cases, the PV starts with more than one coupon. That is why adding only one δ to t_{max} is a little faster as shown in Figure 4.7.

4.3 Approximate Version of TDS+

$\delta = 1$	1s	3s	10s	30s	100s
TDS	85	165	188	214	246
TDS+	**178**	**195**	**240**	**261**	**305**
$\delta = 1/2$	1s	3s	10s	30s	100s
TDS	84	143	165	180	197
TDS+	**156**	**171**	**190**	**232**	**257**
$\delta = 1/4$	1s	3s	10s	30s	100s
TDS	79	95	131	156	170
TDS+	**116**	**155**	**175**	**195**	**239**
$\delta = 1/8$	1s	3s	10s	30s	100s
TDS	45	78	85	102	130
TDS+	**102**	**146**	**164**	**180**	**194**

Table 4.1: Coverage for approximate TDS and TDS+..

Two experiments compare the approximate versions of TDS and TDS+. The experiment in Table 4.1 shows the coverage on the test set for fixed values of δ and fixed time limits. TDS+ finishes substantially more test cases than TDS for each tested combination of δ and time limit.

Note that the data in Table 4.1 is the number of problems solved over time. In the limit, both algorithms should solve all 600 problems but it might take an astronomical amount of time. Assume that there is an absolute scale of difficulty, and that TDS+ dominates TDS, then there will be uneven relative gains when increasing the time limits, since the difficulty of the 600 test problems is most likely not evenly spaced on this difficulty scale.

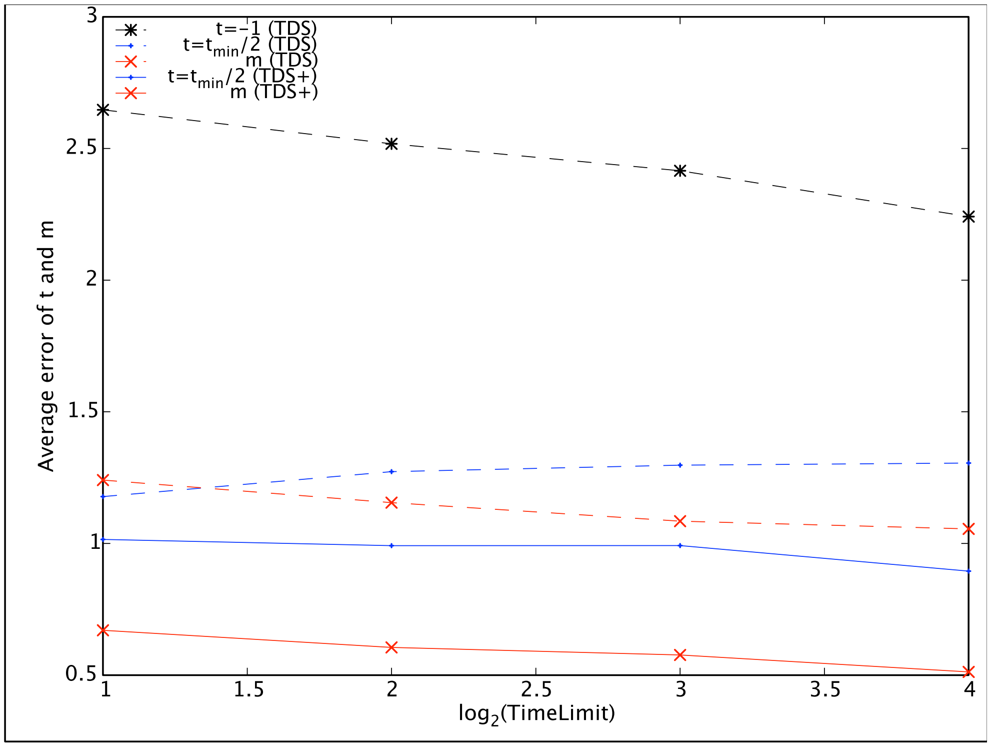


Figure 4.8: Average approximation errors for temperature (t) and mean (m) of TDS and TDS+.

The second experiment measures approximation errors for temperature and mean when varying the time limit. Figure 4.8 shows that both mean and temperature are approximated better by TDS+ than by TDS. If a test case times out, different approximations to the temperature are feasible based on the PV of the incomplete search. Let t_{min} be the minimum coupon value in the PV. As in TDS, if the PV contains a board move, the value of the coupon previous to the first board move is chosen as the estimated temperature. In case there is no board move in the PV, eight different approximations for the temperature were tried: t_{min}, $t_{min} - \delta$, 0, -1, $\frac{t_{min}}{2}$, $\frac{t_{min}-\delta}{2}$, $\frac{t_{min}+1}{2}$ and $\frac{t_{min}-\delta+1}{2}$.

Results are shown in Figure 4.8: a good choice is half the minimum coupon value in the PV, $t = \frac{t_{min}}{2}$. The choice of $t = -1$ used in the original TDS is shown to be poor.

More specifically, the results in Table 4.2 show the average errors for all eight approximations with different time limits using TDS and TDS+. Choosing a value around $\frac{t_{max}}{2}$ is more promising for TDS+ and TDS.

	Time Limit	2	4	8	16
t_{min}	TDS	3.278	2.889	2.556	2.267
	TDS+	1.232	1.115	1.051	0.908
$t_{min} - \delta$	TDS	2.697	2.407	2.127	1.908
	TDS+	1.223	1.109	1.047	0.906
0	TDS	2.060	2.021	1.977	1.876
	TDS+	1.153	1.091	1.070	0.940
-1	TDS	2.647	2.517	2.415	2.241
	TDS+	1.301	1.194	1.149	0.992
$\frac{t_{min}}{2}$	TDS	1.178	1.273	1.298	1.306
	TDS+	**1.015**	**0.992**	0.992	0.895
$\frac{t_{min}-\delta}{2}$	TDS	1.364	1.400	1.396	1.373
	TDS+	1.049	1.003	0.990	**0.887**
$\frac{t_{min}+1}{2}$	TDS	1.070	1.206	1.243	1.282
	TDS+	**1.015**	**0.992**	0.993	0.896
$\frac{t_{min}-\delta+1}{2}$	TDS	1.178	1.273	1.298	1.306
	TDS+	1.046	1.001	**0.989**	**0.887**

Table 4.2: Average t errors using different approximations in TDS and TDS+.

4.4 Using TDS+ to Check if $t(G) > T$ for a Given Temperature Threshold T

The MTS algorithm of [13] can be efficient, but requires recognizing and evaluating the mean of all positions of nonpositive temperature. MTS can be made into a general algorithm by using TDS+ as a temperature threshold checker. For any position MTS encounters in the search, first use TDS+ to check if the current position G is hot or not, i.e., whether the temperature of the G is greater than 0 or not. If G is hot, it satisfies the condition of MTS to keep searching. If G is not hot, then treat it as a terminal position for MTS and evaluate it by the mean calculated by TDS+. The current experiment checks whether TDS+ is suitable as such a leaf node recognizer for MTS. While only testing $t(G) > 0$ is needed in MTS, experiments using

TDS+ on the same test set as the previous section check $t(G) > T$ for all thresholds $T \in \{0,1,2\}$. This could be useful for developing approximate versions of MTS for complex games.

	$t_{\text{true}} \leq T$	$t_{\text{true}} > T$
$t_{\text{test}} \leq T$	low-low	fail-low
$t_{\text{test}} > T$	fail-high	high-high

Table 4.3: Possible results for TDS+ to check if $t(G) > T$, for threshold T.

In Table 4.3, t_{test} is the temperature estimate returned by TDS+ and t_{true} is the true temperature for G. Only the errors for low-low and fail-low situations are calculated because if TDS+ thinks the temperature of G is greater than the threshold T, which corresponds to the situation that G is hot in MTS, it keeps searching without needing to know the exact temperature and mean. Notice that in the experiments, fail-low situations never happen because TDS+ always lowers the temperature in the range from t to $\hat{t}$.

$T = 0$	1s	3s	10s	30s	100s
avg m errors	0.052	0	0	0	0
max m errors	0.5	0	0	0	0
#fail-low	5	0	0	0	0
$T = 1$	1s	3s	10s	30s	100s
avg m errors	0.073	0.034	0.034	0.03	0.03
max m errors	0.5	0.25	0.25	0.25	0.25
#fail-low	25	19	18	12	12
$T = 2$	1s	3s	10s	30s	100s
avg m errors	0.108	0.093	0.093	0.045	0.044
max m errors	0.875	0.875	0.875	0.453	0.375
#fail-low	10	10	10	10	7

Table 4.4: Coverage for TDS+ to check $t(G) > T$.

Results in Table 4.4 show that when the time limit increases, the average and maximum errors for mean and the number of fail-low cases all decrease.

Since both the average and maximum errors for the mean are small, TDS+ seems to be a good tool which can be used to develop an approximate version of mean-temperature search.

4.5 Comparing Sum Games Players

Four different players were tested in a sum game experiment similar to Table 3 of [19]. A sum of several small Amazons positions is played twice, with colors reversed. Each subgame contains one Amazon of each player, plus random obstacles. As in the experiment mentioned above, results are averaged over 200 runs with different randomly generated subgames where one queen each and three burnt-off squares were placed into each subgame at random locations.

Arrow is a full-board $\alpha\beta$ player [21]. The three other players use Hotstrat [5], but differ in their method for computing temperature estimates. Hotstrat-TDS uses the original TDS algorithm, Hotstrat-TDS+ uses TDS+, and CGDB uses a database with exact temperatures. (See Appendix B for details.) The database CGDB uses is calculated and stored by a generator designed by Markus Enzenberger [28], which is a part of the Arrow [21] code base. The database is only available for the case of 4×4 subgames. Players share the same Amazons-specific code, core $\alpha\beta$ search engine with standard enhancements, and heuristic evaluation function. In the first two experiments of Hotstrat-TDS+ vs Arrow and Hotstrat-TDS+ vs Hotstrat-TDS, the time limit is 10 seconds per move.

Table 4.5, 4.6 and 4.7 show the game results depending on the number N and the size of the subgames. Each entry in these three tables shows the mean score, the standard deviation of the score and the percentage of wins for player A in the result of player A vs player B. Note that only the games with non-zero result value are used to compute the percentage of wins.

N	4×4	5×5	6×6
2	-2.20(6.06) 44.0%	-1.62(8.95) 52.3%	0.58(11.8) 57.3%
4	-2.60(8.49) 49.3%	2.54(12.3) 58.6%	25.4(19.7) 77.5%
6	-1.58(10.1) 50.1%	16.4(16.9) 72.8%	53.9(25.4) 86.8%

Table 4.5: Hotstrat-TDS+ vs Arrow

Table 4.5 shows the result for games between Hotstrat-TDS+ and Arrow. The performance of Hotstrat-TDS+ improves strongly with the size and number of subgames. For 4×4 subgames, full board $\alpha\beta$ is slightly superior, but for the cases with many large subgames, Hotstrat-TDS+ wins big. It is interesting to contrast these results with Table 3 of [19], obtained 10 years ago on much slower hardware. The relative performance of $\alpha\beta$ is much improved for simple subgames due to the extra search depth reached, but the superior scaling of local search remains clear for more complex subgames.

N	4×4	5×5	6×6
2	9.77(5.53) 82.5%	22.2(9.26) 88.2%	43.2(13.5) 91.4%
4	19.7(7.59) 90.0%	39.7(12.2) 94.7%	61.1(16.2) 97.7%
6	29.9(9.85) 93.3%	50.7(15.5) 96.7%	76.6(21.6) 98.8%

Table 4.6: Hotstrat-TDS+ vs Hotstrat-TDS.

Table 4.6 matches Hotstrat-TDS+ against Hotstrat-TDS. The experimental setting is the same as in Table 4.5. Hotstrat-TDS+ performs much better. Both methods are based on approximate temperatures, but TDS+ can compute better approximations in the same time, as also seen in Figure 4.8.

Time	$2 \times 4 \times 4$	$4 \times 4 \times 4$	$6 \times 4 \times 4$
1	-2.86(4.99) 35.64%	-8.17(7.75) 27.04%	-15.95(8.61) 15.46%
3	-1.51(5.81) 42.93%	-7.01(7.84) 30.89%	-12.30(9.29) 23.33%
10	-1.19(6.02) 44.63%	-6.43(7.51) 29.22%	-7.82(9.77) 29.46%
30	-1.16(5.59) 46.09%	-3.58(7.79) 40.27%	-5.62(9.18) 35.43%
100	-1.13(6.01) 46.29%	-3.76(7.42) 38.14%	-5.69(9.37) 33.87%

Table 4.7: Hotstrat-TDS+ vs CGDB.

The results of Table 4.7 show that Hotstrat-TDS+ is weaker than the CGDB player with its perfect temperature databases. When the time limit increases, the score difference decreases gradually and the percentage of wins increases. This indicates that TDS+ temperature approximation works effectively and can approach the true temperatures in the limit.

Chapter 5

Other Related Work

5.1 Other Sum Game Algorithms

In [32], Yedwab compared some strategies on playing well in sum games: “**follow the leader**” [18] where a player only plays locally optimal moves and plays in the same game as the opponent whenever possible, “**mean strategy**” [11] where a player always plays t-optimal moves and will have a error bound of the maximum temperature of subgame G_i in the final game value, and “**thermostratic strategy**” [5] based on thermographs. She also improved the error bound in “mean strategy” to be the second highest temperature among all subgames G_i.

5.2 Other Approaches for Computing Amazons

Song [28] created an $\alpha\beta$ player which uses the endgame databases like in Tegos [30], and improved the bounds heuristics for Amazons. He solved the 5×6 Amazons starting position as a first player win.

Kloetzer [15] compared different algorithms, $\alpha\beta$, Monte Carlo Tree Search (MCTS) and temperature discovery search, on playing Amazons. He showed that traditional proof number search is best suited for the task of finding moves in a subgame, and a MCTS player performs better in sum games when the number of subgames is large.

Chapter 6

Future Work

6.1 Combining MTS and TDS+

Since Kao's Mean and Temperature Search (MTS) cannot deal with nonpositive-temperature positions and TDS+ can quickly determine whether a position is hot or not, a general algorithm could be developed which would combine a top-level MTS with a TDS+ based hotness checker for identifying leaf nodes of MTS. The result of the comparison between TDS+ and MTS with TDS+ checker in different games would be interesting. It is also interesting to know how TDS+ compares to MTS in games where all nonterminal positions are hot.

6.2 Better Global and Local Search Algorithms Using TDS+

Hybrid algorithms as discussed in Section 2.3.3 can be developed for sum games. Such algorithms combine local temperature estimates with shallow global search as in [20]. In the experiments of that paper, the algorithm used for playing sum games is only a hotstrat player. Many options for creating a good sum games player are available, such as first generating several potential good moves in several subgames according to the estimated temperatures of the subgames, then using global search only on these moves, pruning moves in low temperature subgames, and those that are locally unpromising according to a modified TDS+ algorithm.

6.3 Proof of the 2δ-Conjecture

In all the experiments conducted in this paper, the 2δ-Conjecture never failed from Section 3.3.2. It is still important to prove its correctness or find a counterexample.

6.4 Relation Between Searches With Different δ

A more general question concerns the relation between the search information computed by different searches using different values of δ.

Is it possible to derive bounds from these searches? One idea is that for the search using some δ, if temperatures of subpositions during this search are all even multiples of δ, we can assume or assure that this δ is already small enough, thus stop decreasing δ immediately and use the results of that search. But it still needs a proof and testing.

In TDS+, only a single hash table is used to store all the search information with different δs. Right now, this information is not exploited effectively, since only approximate temperature is used in the algorithm and all other search information in the hash table is ignored afterwards. If more information from the searches with larger δs can be exploited, then searches with smaller δ could be sped up more.

6.5 Proof for the Exact Temperature Set for Short Games Born By day n

Only a proof for the superset of the possible temperatures for short games born by day n is provided in Appendix A. If the exact possible temperature set for short games born by day n can be determined, E_2 can skip more impossible temperatures, which leads to a faster search.

6.6 Applications to Other Combinatorial Games

Testing TDS+ on other combinatorial games, like NoGo and Go, would be interesting. It will be great if TDS+ can be extended to Go endgames. The main technical difficulty here is dealing with local position repetitions called *ko*.

Bibliography

[1] M. Albert, R. Nowakowski, and D. Wolfe. *Lessons in play: an introduction to combinatorial game theory*. A K Peters, Wellesley, 2007.

[2] B. Arneson, R.B. Hayward, and P. Henderson. Monte Carlo Tree Search in Hex. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):251–258, 2010.

[3] E. Berlekamp. The economist's view of combinatorial games. In R. Nowakowski, editor, *Games of No Chance: Combinatorial Games at MSRI*, pages 365–405. Cambridge University Press, 1996.

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

[5] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways*. Academic Press, London, 1982. Revised version published 2001-2004 by AK Peters.

[6] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[7] J. Conway. *On Numbers and Games*. A K Peters Ltd., 2001.

[8] M. Enzenberger, M. Müller, B. Arneson, and R. Segal. Fuego - An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):259–270, 2010.

[9] T. Furtak, M. Kiyomi, T. Uno, and M. Buro. Generalized Amazons is PSPACE-complete. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 132–137. Morgan Kaufmann Publishers Inc., 2005.

[10] S. Gelly and D. Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[11] O. Hanner et al. Mean play of sums of positional games. *Pacific J. Math*, 9:81–99, 1959.

[12] K. Kao. Mean and temperature search for Go endgames. *Information Sciences*, 122(1):77–90, 2000.

[13] K. Kao, I. Wu, Y. Shan, and S. Yen. Selection search for mean and temperature of multi-branch combinatorial games. *ICGA Journal*, 35(3):157–176, 2012.

[14] A. Kishimoto and M. Müller. A general solution to the graph history interaction problem. In *Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 644–649, San Jose, CA, 2004.

[15] J. Kloetzer, H. Iida, and B. Bouzy. Playing Amazons endgames. *ICGA Journal*, 32(3):140–148, 2009.

[16] D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6:293–326, 1975.

[17] R.J. Lorentz. Amazons discover Monte-Carlo. In *Computers and Games*, pages 13–24. Springer, 2008.

[18] J. Milnor. Sums of positional games. *Ann. of Math. Stud.(Contributions to the Theory of Games, HW Kuhn and AW Tucker, eds.), Princeton*, 2(28):291–301, 1953.

[19] M. Müller, M. Enzenberger, and J. Schaeffer. Temperature discovery search. In *Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, pages 658–663, San Jose, CA, 2004.

[20] M. Müller and Z. Li. Locally informed global search for sums of combinatorial games. In J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computers and Games: 4th International Conference, CG 2004*, volume 3846 of *Lecture Notes in Computer Science*, pages 273–284, Ramat-Gan, Israel, 2006. Springer.

[21] M. Müller and T. Tegos. Experiments in computer Amazons. *More Games of No Chance*, 42:243–260, 2002.

[22] Y. Sato, D. Takahashi, and R. Grimbergen. A shogi program based on Monte-Carlo tree search. *ICGA Journal*, 33(2):80–92, 2010.

[23] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[24] J. Schaeffer, R. Lake, P. Lu, and M. Bryant. CHINOOK the world man-machine checkers champion. *AI Magazine*, 17(1):21, 1996.

[25] J. Schaeffer and A. Plaat. Kasparov versus Deep Blue: The rematch. *ICCA Journal*, 20(2):95–101, 1997.

[26] A. Siegel. *Combinatorial Game Theory*, volume 146 of *Graduate Studies in Mathematics*. American Mathematical Society, 2013.

[27] R.G. Snatzke. Exhaustive search in the game Amazons. *More Games of No Chance*, 42:261–278, 2002.

[28] J. Song. An Enhanced Solver for the Game of Amazons. Master's thesis, University of Alberta, 2013.

[29] J. Song and M. Müller. An Enhanced Solver for the Game of Amazons. *To Appear in TCIAIG*, 2014.

[30] T. Tegos. Shooting the last arrow. Master's thesis, University of Alberta, 2002.

[31] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[32] L. Yedwab. On playing well in a sum of games. Master's thesis, Massachusetts Institute of Technology, 1985.

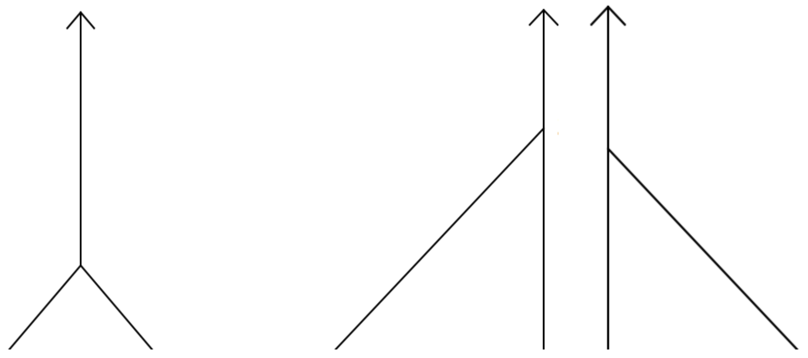
Appendix A

Possible Means and Temperatures

In this Appendix, first characterizes the sets T_n and M_n that contain all possible temperatures and means of games born by day n. Next, an example show that T_n and M_n are not exact sets.

A.1 For Games Born by Day n

Theorem 1. *For a game born by day $n \in \mathbb{N}$, its temperature is contained in the set $T_n = \{-\frac{1}{2^b}, 0, \frac{1}{2^b}, \frac{3}{2^b}, \dots, a + \frac{1}{2^b} | 0 \le a \le n-2, 0 \le b \le n-1\}$, and its mean in the set $M_n = \{0, \pm\frac{1}{2^b}, \pm\frac{3}{2^b}, \dots, \pm(a + \frac{1}{2^b}), \pm n | 0 \le a \le n-2, 0 \le b \le n-1\}$.*



(a) both taxed (b) left taxed (c) right taxed

Figure A.1: Different mast intersections.

Intuition: Mean and temperature can be determined by the mast value and its corresponding temperature directly. Figure A.1 shows the three kinds of possible mast intersections. We only need to consider two cases since (b) and (c) are symmetric.

If we start from a position G with L moving first, the sequence of positions visited will be $G^L, G^{LR}, G^{LRL}, \ldots$. These positions are called the left alternating followers of G, denoted by $G^{L(1)}, G^{L(2)}, G^{L(3)}, \ldots$. An alternating follower of G is called odd (even) if it can be reached from G by an odd (even) number of moves.

For mast (a), the left wall and right wall are determined by some left odd option $G^{L(x)}$ and right odd option $G^{R(y)}$ respectively where x and y are both odd.

For mast (b), the left wall and right wall are determined by some left odd option $G^{L(x)}$ and right even option $G^{R(y)}$ respectively where x is odd and y is even.

Now we begin the proof for the theorem:

Proof. First, by Proposition 5.20(a) in (Siegel 2013), G is cold (temperature is less than zero) if and only if G is equal to a number. A number can be represented as $x = \frac{m}{2^n}$, and $t(x) = -1/2^n$.

A game G born on day n or less has mean and temperature of the form $x/2^{n-1}$ with integer x. This follows easily by induction from adapting the proof of [7, Theorem 61].

Combining the results above, if G is born on day n and its temperature is negative, then $t(G) \in \{-\frac{1}{2^b} | 0 \leq b \leq n-1\}$ Also on day 1, 0 is one of the new temperatures. Thus we only need to prove the positive part of T_n.

We use induction to prove T_n and M_n. When $n = 0$, $T_n = \{-1\}, M_n = \{0\}$, the theorem holds. Now we assume it holds for any positive integer up to n. We need to prove that it still holds for $n+1$.

Let T_n^* and M_n^* be the exact temperature set and mean set for all games born by day n. We need to prove $T_{n+1}^* \subseteq T_{n+1}$, $M_{n+1}^* \subseteq M_{n+1}$.

In this part, we try to get the maximum possible temperature, and the maximum and minimum possible mean of game G born on day $n+1$.

When we draw the thermograph of some game G, there are only 3 cases for the mast of the thermograph.

In the case in Figure A.1a the mast base point of game G comes from the intersection of the taxed masts of some left option $G^{L(x)}$ and some right option $G^{R(y)}$

where x and y are both odd. Therefore we have:

$$t(G) = \frac{m(G^{L(x)}) - m(G^{R(y)})}{2}$$

$$m(G) = \frac{m(G^{L(x)}) + m(G^{R(y)})}{2}$$

So $t(G) \leq \frac{\sup(m(G^{L(x)})) - \inf(m(G^{R(y)}))}{2} = \frac{n-(-n)}{2} = n$ since $G^{L(x)}$ and $G^{R(y)}$ are born by day n.

Similarly, $m(G) \leq \frac{\sup(m(G^{L(x)})) + \sup(m(G^{R(y)}))}{2} = \frac{n+n}{2} = n$
and $m(G) \geq \frac{\inf(m(G^{L(x)})) + \inf(m(G^{R(y)}))}{2} = \frac{-n-n}{2} = -n$.

Then consider two cases when the mast intersection G comes from one vertical and one diagonal mast. In Figure A.1b, the mast of G comes from some left option $G^{L(x)}$ and some right option $G^{R(y)}$ where x is odd and y is even. In this case, $t(G) \leq t(G^{R(y)}) \leq n-1$ and $-n \leq m(G) = m(G^{R(y)}) \leq n$. The third case in Figure A.1c is symmetric.

In the special case when G^L or G^R is empty, $t(G) = -1 \in T_{n+1}$ and $m(G) = m(G^L)+1$ and $m(G) = m(G^R)-1$ respectively, thus $m(G) \in M_{n+1}$ since $\sup(m(G^L))+1 = n+1$ and $\inf(m(G^R))-1 = -(n+1)$ are both in M_{n+1}.

Combining the results above, $t(G) \leq n+1$ for game G born on day $n+1$. Since T_{n+1} contains all the possible temperatures less than or equal to n and M_{n+1} contains all the possible means less than or equal to $n+1$, $T^*_{n+1} \subseteq T_{n+1}$ and $M^*_{n+1} \subseteq M_{n+1}$.

$\square$

A.2 An Example Why $T_n \neq T_n^*$ and $M_n \neq M_n^*$

$t \backslash m$	0	$\pm\frac{1}{8}$	$\pm\frac{1}{4}$	$\pm\frac{3}{8}$	$\pm\frac{1}{2}$	$\pm\frac{5}{8}$	$\pm\frac{3}{4}$	$\pm\frac{7}{8}$	± 1	$\pm\frac{9}{8}$	$\pm\frac{5}{4}$	$\pm\frac{11}{8}$	$\pm\frac{3}{2}$	$\pm\frac{13}{8}$	$\pm\frac{7}{4}$	$\pm\frac{15}{8}$	± 2	$\pm\frac{17}{8}$	$\pm\frac{9}{4}$	$\pm\frac{5}{2}$
$\frac{1}{8}$		√		√		√		√												
$\frac{1}{4}$	√		√		√		√		√		√				√					
$\frac{3}{8}$		√		√		√				√										
$\frac{1}{2}$	√		√		√				√				√							√
$\frac{5}{8}$		√		√				√				√								
$\frac{3}{4}$	√		√		√		√				√								√	
$\frac{7}{8}$		√		√		√				√								√		
1	√		√		√				√								√			
$\frac{9}{8}$		√		√				√								√				
$\frac{5}{4}$	√		√				√								√					
$\frac{11}{8}$		√				√								√						
$\frac{3}{2}$	√				√								√							
$\frac{13}{8}$				√								√								
$\frac{7}{4}$			√								√									
$\frac{15}{8}$										√										
2	√								√											
$\frac{17}{8}$								√												
$\frac{9}{4}$							√													
$\frac{5}{2}$					√															
3	√																			

Table A.1: All possible (t,m) pairs for $t > 0$ of games born by day 4.

Table A.1 shows all possible (t,m) pairs of games born by day 4 when $t > 0$. All temperatures in T_4 do occur. However, for games born on day 5, $\frac{43}{16}$ is contained in both sets M_5 and T_5, but no game with birthday 5 can be constructed with $\frac{47}{16}$ as its mean or temperature. If we have such a game G and $m(G) = \frac{47}{16} \in M_5^*$, it has to be in the case of Figure A.1a due to its denominator. The only choice is using $M(G^L) = 4$ and $M(G^R) = \frac{15}{8}$ as the two options whose masts intersect in the thermograph of G. From Table A.1 we know that for game G^R whose $m(G^R) = \frac{15}{8}$, $t(G^R)$ can only be $\frac{9}{8}$ but here if we use $M(G^L) = 4$ and $M(G^R) = \frac{15}{8}$ to construct the game with $m = \frac{47}{16}$ the temperature $t(G)$ must be $\frac{17}{16}$ which is less than $\frac{9}{8}$ and this is impossible to construct a game in Figure A.1a case. Reason for $T_n^* \neq M_n^*$ can be found by using a similar example.

Appendix B

How to Use the TDS+ Software

Our algorithms and experiments are contained in the project called *Arrow* [21] which is an Amazons player. Arrow is built on top of the Fuego framework [8]. Project Arrow can be used to load Amazons positions and make Amazons moves using several different algorithms such as $\alpha\beta$ search, MCTS, or sum game players.

To test our algorithm TDS+ on Amazons positions, we use a another project called *CGT* in which our core algorithm lies. The search engine is called *CpSearch*.

We used the basic $\alpha\beta$ search engine, *SgSearch*, from project *Fuego* [8]. *CpSearch* is a subclass of *SgSearch*.

B.1 Exact TDS

The flow of determining the temperature and mean of a game G is as follows:

- First, Arrow loads an Amazons test case G and starts *CpSearch*.
- Then, *CpSearch* searches in $G+C$, where C is a coupon stack. It calls the functions in *SgSearch* but uses its own move-generation function and modified hash tables.
- Finally, $m(G)$ and $t(G)$ are determined from the value of the searches and the move sequences in the principal variations as reviewed in Section 2.4.3.

To test TDS+ in a new game, we first need to have a project to include the game playing functions of the new game: loading a board, board move generation, board evaluation function etc., then follow the same process above.

B.2 Experiments

The Amazons test cases were randomly chosen from the positions generated by CGDB which was used to create the databases for Amazons. First the size of a rectangle containing all database positions is chosen, in our case 4×4. CGDB starts generating all positions with zero empty squares, whose birthday is 0. Then it generates all positions with more squares and computes the thermographs for them by using all results of positions born on earlier days. All symmetric positions are considered the same position and only one of them is kept for randomly generating the test set. Each enhancement in our algorithm can be switched on or off manually during the experiments. We run a regression test suite called *"reg-test-tds"* on all these test cases using different versions of TDS. This test suite is now part of the Arrow code base in *"project_arrow/regression/reg-test-tds"*.

Sum game players use an approximate version of TDS+ or TDS using a test suite called *"experiments-splitboard-games"* which is also a part of the Arrow code base. The only difference between approximate and exact versions of TDS+ is that the approximate version will determine the temperature and mean approximately by the information of the searches finished in a limited time.

Different sum game players were tested in Section 4.5. The first two are hotstrat players called Hotstrat-TDS/TDS+. They first use TDS/TDS+ to estimate the temperatures for all subgames, then select a move with the second highest temperature [19]. *Arrow* is a sum game player which uses global $\alpha\beta$ search with the evaluation function from [21]. The CGDB player is also a hotstrat sum game player. Instead of using TDS/TDS+ to compute the temperature, it uses a pre-calculated database of all 4×4 positions which stores all information including temperature, mean and best move of each color for each position.