TDS+: Improving Temperature Discovery Search

Yeqin Zhang and **Martin Müller**
Computing Science, University of Alberta
Edmonton, Canada
{yeqin,mmueller}@ualberta.ca

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.

Introduction

Two player games with perfect information can be analyzed by search techniques based on the minimax principle. Standard forward search techniques include alphabeta ($\alpha\beta$) search (Knuth and Moore 1975), proof number search and its many variants (Kishimoto et al. 2012), and recently, Monte Carlo Tree Search (Browne et al. 2012). Retrograde analysis (Bellman 1965) constructs endgame databases by backwards search. Such databases reduce the forward search depth needed to solve a game.

Many games of small to medium complexity have been solved by such search methods. Effective techniques for more complex games employ deep search which uses either a game-specific evaluation function, or randomized simulations, or a combination. The focus of the work presented here is on games with special combinatorial structure, which allows powerful purely algorithmic improvements.

Combinatorial Game Theory

Combinatorial game theory (Berlekamp, Conway, and Guy 1982; Conway 2001) studies games which 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 (Siegel 2013).

The ancient Asian game of Go is a very well-studied classical two player game. Go endgames can be mapped to the combinatorial game model. This mapping is meaningful but not quite perfect because of problems related to position repetition rules (Berlekamp and Wolfe 1994; Müller 1999). Like much prior work, the examples and experiments here use the game of Amazons, which combines features of both chess and Go and is a perfect fit for the sum game model.

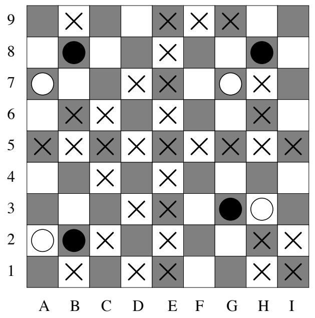

Figure 1: *Amazons* position with independent subgames in each corner, from (Müller, Enzenberger, and Schaeffer 2004).

In Amazons, each player controls four amazons which move like chess queens, but also shoot arrows which "burn off" empty squares. An Amazons move is specified by the locations involved as **from** $-$ **to** $\times$ **arrow**, such as move **A2–A3**$\times$**C3** for White in Figure 1. Areas that are completely surrounded by burned off squares can be viewed as subgames which are independent from the rest of the board. In Figure 1, from (Müller, Enzenberger, and Schaeffer 2004), solid walls of burnt off squares separate a roughly 4×4 region in each corner to form a subgame that is independent from the rest of the board. In terms of combinatorial game theory, the overall game position can be represented as a *sum* of four much simpler subgames $G_1 + G_2 + G_3 + G_4$.

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. Two important characteristics of a subgame are its *temperature*, which measures the urgency of playing in a subgame, and its *mean*, a score indicating the advantage of the Black player in this subgame. By convention, for scoring purposes, Black = Left = positive and White = Right = negative.

Classically, means and temperatures are computed bottom-up from terminal positions, using the *thermograph* structure, which contains enough information to determine both these measures (Berlekamp, Conway, and Guy 1982).

Two forward search approaches for computing means and temperatures are known. *Mean and Temperature Search* (MTS) (Kao 2000; Kao et al. 2012) searches a subgame in 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. The remainder of this paper focuses on improving the other known forward search algorithm, *Temperature Discovery Search* (TDS) (Müller, Enzenberger, and Schaeffer 2004), which can handle any loopfree game including those with nonpositive temperature.

Enriched Environment, Coupon Stack and Temperature Discovery Search (TDS)

Local analysis of a single subgame can utilize an *enriched environment* (Berlekamp 1996; 2000) 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 $-v$ by the convention above. The mean of such a switch is $\mu(v| - v) = 0$ and its temperature is $t(v| - v) = v$ (Berlekamp, Conway, and Guy 1982). Sum games consisting only of such switches are very easy - playing a switch with highest temperature v is always optimal. Following (Berlekamp 1996), 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 (Berlekamp 1996).

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 $\mu(C) = 0$. For any $t_{max} \geq -1$, the *left score*, the minimax score with alternating play and Left going first, is $V(C, \mathit{Left}) = \lceil \frac{n}{2} \rceil \delta$ while the *right score* with Right going first is $V(C, \mathit{Right}) = -\lceil \frac{n}{2} \rceil \delta$.

Temperature Discovery Search (TDS) (Müller, Enzenberger, and Schaeffer 2004) 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.

Approximate TDS uses a larger value of δ than required by theory. 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 $S(g, c, \Delta, \mathit{toPlay})$, 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.

Motivating Example for Improving TDS

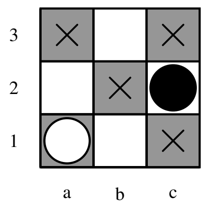

Figure 2: Amazons example where TDS scales badly for small δ.

The main problem of applying the original TDS algorithm in practice is its time complexity when scaling up - either to larger subgames, or higher precision using smaller δ. As an illustration, consider the game G in Figure 2 with temperature $t(G) = \frac{5}{4}$. 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×B1 5.** $C(\frac{2}{2})$
6. $C(\frac{1}{2})$ **7.** $C(0)$ **8.** $C(-\frac{1}{2})$ **9. C2–B3×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[1]. However, computing the exact temperature requires setting $\delta = 1/8$. This leads to a deep 23-ply search with the following PV:

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**.

This optimal line of play contains two long coupon-taking subsequences but only two moves (8 and 23) on the game

[1]For simpler presentation, we ignore the mechanism for handling *sente* moves in this example. For details, see (Müller, Enzenberger, and Schaeffer 2004).

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×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.

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 the following section achieves this *while retaining correctness*.

The TDS+ Algorithm: Speeding up TDS

The approaches to improving TDS in this paper are based on three main insights: 1. 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 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.

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 are rules for handling 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 (Berlekamp, Conway, and Guy 1982; Müller, Enzenberger, and Schaeffer 2004). 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 agressive use of tables in TDS+ requires handling the GHI problem above. While efficient general solutions exist (Kishimoto and Müller 2004), 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 $S(G', c_{-1}, \Delta, Black, 1)$.

Conditional Move Generation

Algorithm 1 Conditional Move Generation using a temperature-dependent Skip() test.

```
1: function CONDITIONALGENERATE($G$, $C$, Skip)
2:     $t$ := MaxTemperature($C$)
3:     if Skip(t, $G$) then
4:         return $\{C(t)\}$
5:     else
6:         return $\{C(t)\} \cup$ Generate($G$)
7:     end if
8: end function
```

Several 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 1, a uniform δ 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 Five Enhancements of TDS+

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

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 $n - 1$. However, the temperature of most positions is much lower. As for Figure 2, 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**: 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 $t(G) \leq t_\delta + 2\delta$.

Algorithm 2 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 5 of the algorithm could possibly fail. In this case, the algorithm needs to re-search with $t_{max} = t_\delta + 3\delta$, $t_{max} = t_\delta + 4\delta$, etc. until it succeeds. This case has never happened in thousands of experiments, and is not shown in the pseudocode. 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 2 TDS$_1$: Pre-searches with δ from 1 to 2^{-n}

```
1:  function TDS_1(G, n)
2:      δ := 1
3:      t_max := safe_bound(G)              ▷ n − 1 in Amazons
4:      while δ > 2^{−n} do
5:          t_δ := TDS(G, δ, t_max)
6:          t_max := t_δ + 2δ    ▷ Adapt t_max for next iteration
7:          δ := δ/2
8:      end while
9:      return TDS(G, δ, t_max)
10: end function
```

E$_2$: Avoid Search at Impossible Temperatures

The *birthday* $b(G)$ of a loopfree game G is a measure for its recursive depth (Siegel 2013). G is said to be born *by* day n if $b(G) \leq n$. For an Amazons position G with n empty squares, $b(G) \leq n$. For fixed n, the set of games born by day n is finite. In unpublished work, the authors recently proved the following theorem which restricts the sets of possible temperatures and means of loopfree games born by day n.

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

Algorithm 3 Skipping Impossible Temperatures

```
1: function SKIP-IMPOSSIBLE-T(t, G)
2:     n := BirthdayUpperBound(G)
3:     return t ∉ T_n
4: end function
```

Enhancement 2 directly applies this theorem using the SKIP-IMPOSSIBLE-T function in Algorithm 3 as the argument *Skip* in CONDITIONALGENERATE of Algorithm 1. This test requires an estimate of the birthday of a game G. Especially for small δ, many temperatures can be skipped.

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. 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, and this value 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 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.

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. Enhancement E$_4$, shown in Algorithm 4, 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.

Algorithm 4 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
```

E$_5$: Improved Handling of PTP States

With E_5, PTP states reached after two consecutive -1 coupons are recognized as solved positions.

Experiments

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.

Improvement from Individual Enhancements

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. 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. These were randomly sampled from the database. Experiments were performed to test interesting subsets of enhancements, including each enhancement in isolation, as well as a *leave-one-out* setting. Table 1 shows the results in terms of *coverage*, or number of problems solved, with different time limits.

Time	1s	2.5s	10s	25s	100s	250s
TDS	69	73	73	74	76	76
TDS_1	78	88	100	107	116	116
TDS_2	69	73	73	75	76	76
TDS_3	73	76	85	95	117	117
TDS_{134}	**81**	**108**	**129**	**135**	**137**	**141**
TDS_{35}	73	76	86	96	117	117
TDS_{13}	99	117	130	131	136	141
TDS_{235}	73	77	91	102	117	117
TDS_{1345}	82	109	129	135	137	141
TDS_{12}	78	89	100	108	116	116
TDS_{1235}	**96**	**118**	130	135	150	157
TDS_{1234}	83	111	**133**	**136**	**153**	157
TDS_{12345}	82	111	**133**	**136**	**153**	**159**

Table 1: Coverage for selected subsets of TDS enhancements. Results for: plain TDS, individual enhancements, TDS_{13}, leave-one-out, and full TDS+. E_4 and E_5 require other enhancements to work.

Discussing the contribution of each enhancement from the results in the table, 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}.

Adding E_4 is a strong improvement over E_3 alone but not over TDS_{13}. TDS_{12345} and TDS_{1235} also have similar coverage. TDS_{12345} can solve more high temperature test cases. Figure 3 shows details. For high temperature test cases with $t(G) \geq 2$, TDS_{12345} is always faster than TDS_{1235}. Since not all larger size test cases with 6 or more empty squares have high temperature, TDS_{1235} can finish some of them faster than TDS_{12345}.

Figure 4 gives further evidence that reducing the number of top coupons as in E_1 is important. In this experiment, TDS^{+n}_{135} is TDS_{135} but with n extra top coupons of value $t_{max}+\delta, \cdots, t_{max}+n\delta$ added after determining t_{max} with enhancement E_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 $n = 1 \ldots 4$. Runtimes are plotted on a log scale in Figure 3 and 4.

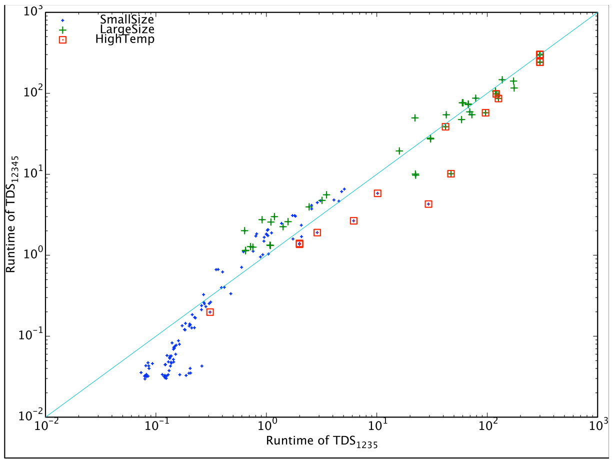

Figure 3: Runtime comparison, TDS_{1235} vs TDS_{12345}

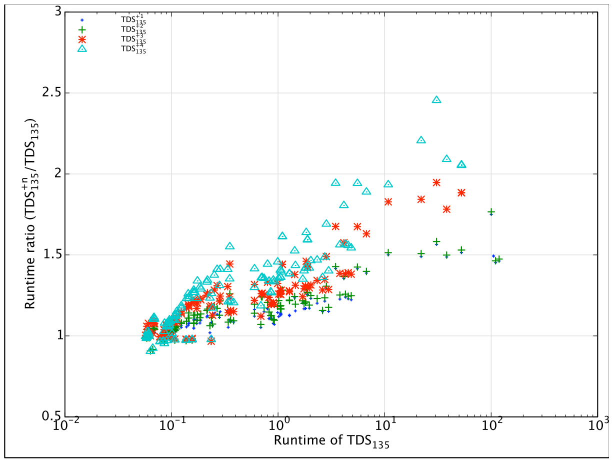

Figure 4: Runtime comparison, TDS_{135} vs. TDS^{+n}_{135}

The improvement from E_5 is modest. As an example, TDS_{12345} expands 0.5% fewer nodes than TDS_{1234} over the set of all 50 test cases with 5 empty squares.

Approximation Error of TDS+ with Larger δ

Two experiments compare the approximate version of TDS and TDS+. The first experiment in Table 2 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 difference in coverage fluctuates a bit, as clusters of problems with similar difficulty can occur.

The second experiment measures approximation errors for temperature and mean when varying the time limit. Figure 5 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. As in TDS, if the PV contains a board move, the prior coupon value is chosen as the temperature. In case there is no board move

$\delta = 1$	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 2: Coverage for approximate TDS and TDS+.

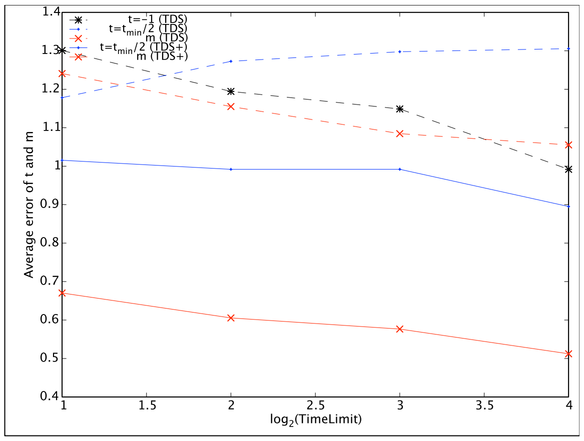

Figure 5: Approximation errors for temperature (t) and mean (m) of TDS and TDS+.

in PV, eight different approximations for the temperature were tried. Only selected results are shown in Figure 5: a good choice is half the minimum coupon value in the PV, $t = t_{min}/2$. The choice of $t = -1$ in the original TDS is poor.

Comparing Sum Game Players

Four different players were tested in a sum game experiment similar to Table 3 of (Müller, Enzenberger, and Schaeffer 2004). A sum of several small Amazons positions is played twice, with colors reversed. Each subgame contains one Amazon of each player, plus some 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. The three other players use Hotstrat (Berlekamp, Conway, and Guy 1982), 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. 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, the time limit is 10 seconds per move.

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 3: Game results depending on the number N and the size of the subgames. Each entry shows the mean score, the standard deviation of the score and the percentage of wins for Hotstrat-TDS+ vs Arrow.

Table 3 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 (Müller, Enzenberger, and Schaeffer 2004), 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: Hotstrat-TDS+ vs Hotstrat-TDS.

Table 4 matches Hotstrat-TDS+ against Hotstrat-TDS. The experimental setting is the same as the setting in Table 3. Hotstrat-TDS+ performs much better. Both methods are based on approximate temperatures, but TDS+ can compute better approximations in the same time, as was seen in Figure 5.

Details of the matches between CGDB vs Hotstrat-TDS+ for 4×4 with $N = 2, 4, 6$ are omitted for lack of space. This result shows that Hotstrat-TDS+ is able to find good-enough moves to compete with the perfect temperature knowledge from the database when the time limit increases.

Conclusions and Future Work

TDS+ contains algorithmic enhancements that greatly speed up temperature discovery search in both its exact and heuristic versions at no cost to precision. Some of the enhancements such as E_4 promise to scale well for even larger subgames.

Future work includes: 1. settle the status of the 2δ-Conjecture. 2. use TDS+ to generalize Kao's Mean and Temperature Search (MTS). A hybrid algorithm would combine a top-level MTS with a TDS+ based hotness checker for identifying leaf nodes of MTS. 3. extend TDS+ to Go endgames. The main technical difficulty here is dealing with local position repetitions called *ko*. 4. develop hybrid algorithms that combine local temperature estimates with shallow global search as in (Müller and Li 2006).

References

Bellman, R. 1965. On the application of dynamic programing to the determination of optimal play in chess and checkers. *Proceedings of the National Academy of Sciences of the United States of America* 53(2):244–247.

Berlekamp, E., and Wolfe, D. 1994. *Mathematical Go: Chilling Gets the Last Point*. Wellesley: A K Peters.

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

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

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

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

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

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

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

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

Kishimoto, A.; Winands, M.; Müller, M.; and Saito, J. 2012. Game-tree search using proof numbers: The first twenty years. *ICGA Journal* 35(3):131–156.

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

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

Müller, M.; Enzenberger, M.; and Schaeffer, J. 2004. Temperature discovery search. In *Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, 658–663.

Müller, M. 1999. Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, 578–583.

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