# Locally Informed Global Search for Sums of Combinatorial Games

Martin Müller and Zhichao Li

Department of Computing Science, University of Alberta
Edmonton, Canada T6G 2E8
mmueller@cs.ualberta.ca, zhichao@ualberta.ca

**Abstract.** There are two complementary approaches to playing sums of combinatorial games. They can be characterized as local analysis and global search. Algorithms from combinatorial game theory such as *Hotstrat* and *Thermostrat* [2] exploit summary information about each subgame such as its *temperature* or its *thermograph*. These algorithms can achieve good play, with a bounded error. Their runtime depends mainly on the complexity of analyzing individual subgames. Most importantly, it is not exponential in global parameters such as the branching factor and the number of moves in the sum game. One problem of these classic combinatorial game algorithms is that they cannot exploit extra available computation time.
A global minimax search method such as $\alpha\beta$ can determine the optimal result of a sum game. However, the solution cost grows exponentially with the total size of the sum game, even if each subgame by itself is simple. Such an approach does not exploit the independence of subgames.
This paper explores combinations of both local and global-level analysis in order to develop *locally informed global search algorithms* for sum games. The algorithms utilize the subgame structure in order to reduce the runtime of a global $\alpha\beta$ search by orders of magnitude. In contrast to methods such as Hotstrat and Thermostrat, the new algorithms exhibit improving solution quality with increasing time limits.
**Key words:** sum games, minimax, combinatorial games, Hotstrat+.

## 1 Introduction

Combinatorial game theory studies games that can be decomposed into a sum of independent subgames [2]. Two of many examples are endgames in the games of Go and Amazons. Classical approaches to game tree search work on the global state, and can not profit from the structure of local subgames. Methods for playing sums of games using concepts from combinatorial game theory, that are based purely on local search, have limited accuracy and cannot achieve globally optimal play in general. This paper studies global search approaches for both optimal and approximate play of sum games that are able to effectively utilize local information. It is organized as follows: Section 2 surveys previous work on sum game algorithms. Section 3 describes several methods for using local information in a global search framework. Section 4 describes a simple model for generating random abstract hot combinatorial games, and uses it to experimentally evaluate and compare seven methods for playing sums of games. Section 5 concludes the paper and lists future work.

## 2 Sum Game Algorithms

Algorithms for playing sum games usually contain both a local and a global component. On the local level, individual subgames are analyzed. On the global level, the results of local analyses are combined into an overall strategy. In terms of running time, usually the cost of game tree search dominates all other factors. Therefore it makes sense to distinguish between algorithms that use only local search, only global search, or a combination of both.

### 2.1 Approximate Combinatorial Game Algorithms that Use Only Local Search

Computing properties of a combinatorial game position such as its canonical form, its thermograph or its temperature, can be viewed as a local search. Typically, the entire game tree must be expanded. Classical combinatorial game algorithms use only local analyses of this form. On the global level, they perform a static analysis. Examples are Thermostrat, Hotstrat, and Sentestrat [2, 10]. All these algorithms play well, but not perfectly. The errors of Thermostrat and Sentestrat can be bounded by the temperature of the hottest or second-hottest subgame [10], while no such bound is known for Hotstrat.

**Hotstrat+** Hotstrat is one of the simplest sum game strategies. It computes the temperature of all subgames, and plays a move in the hottest subgame. A small modification, *Hotstrat+*, performs slightly but consistently better in the experiments reported in this paper. This variant computes a *pseudo-temperature* for each subgame and a given player, defined as the largest $t$ for which the thermograph is equal to its taxed option. For example, in the *sente* position $10|0|| - 1$, the temperature is 1 but Left can already play at $t = 5$, since Left's move to $10|0$ is a threat that raises the temperature of the game to 5. As Left, Hotstrat+ plays in this position at the pseudo-temperature $t = 5$, while plain Hotstrat waits until $t = 1$. As Right, both strategies wait until $t = 1$.

Intuitively, the advantage of Hotstrat+ is that it cashes in free sente moves earlier and reduces the opportunities for the opponent to play "reverse sente" moves such as the move from $10|0|| - 1$ to -1 in the example above.

In this paper, when the meaning is clear from the context, pseudo-temperatures computed by Hotstrat+ will simply be called temperatures.

### 2.2 Decomposition Search

An exact method for solving sum games that uses a local search approach is *decomposition search* [6]. In this method, local searches are used to compute the canonical form of each subgame. From the canonical forms, *incentives* of all moves are derived. An incentive measures the difference between the position before and after a move. The incentive of a move by Left from game $G$ to option $G^L$ is defined as $G^L - G$, while the incentive of a Right move from $G$ to $G^R$ is $G - G^R$. The asymmetric definition implements the idea that larger incentives are always better, for either player. In combinatorial games, by convention Left wants to maximize the result while Right wants to minimize it.

Incentives of moves are combinatorial games and as such are partially ordered [1]. If a single move with dominating incentive exists, it is proven to be optimal. In this case the algorithm is very efficient - it found a move using only local search and comparison of locally computed incentives of moves. The other case is when no single move with dominating incentive exists. In this case, a globally optimal play can be found by a combinatorial summation of all subgames. This approach is feasible if the complexity of computing the sum does not lead to a combinatorial explosion. For example, the cooled values of late stage Go endgames investigated by Berlekamp et al. [3, 8] often add well and sums can be computed in a reasonable time. However, this approach is no longer practical for sums of "hotter" Go positions that occur earlier in the game. The complexity of computing the sum of such games quickly explodes. This is shown experimentally in Section 4.2. Fortunately, finding an optimal move in a sum game can be accomplished much faster than computing the sum.

### 2.3 Local Move Pruning for Global Minimax Search

A different approach to solving sum games is investigated in [7]. Local enhancements for speeding up full board $\alpha\beta$ search are used for solving the same kind of Go endgames as in decomposition search. Several methods for local move pruning are developed and shown to improve the search performance by orders of magnitude, compared to plain global $\alpha\beta$ search. However, for these late endgames, where usually a single dominated move exists, even a global search method with many local search improvements is completely dominated by decomposition search.

### 2.4 Complexity Results

The complexity of solving sums of hot games grows quickly. Wolfe [9], building on previous work by Yedwab [10] and Moews [5], showed that playing sums of simple Go endgames of the form $a||b|c$ is PSPACE-hard. On the other hand, the experimental results of decomposition search indicate that if dominating moves exist, the complexity of solving sums does not need to grow exponentially with the size of the sum. Since there appears to be a large gap between the theoretical worst case and the typical case in games such as Go, it is important to develop algorithms that will perform well in practice.

## 3 Using Local Information in Global Search

This paper investigates practical algorithms for playing sums of games in situations where direct summation is impractical. In this sense, the work reported here can be seen as a natural extension of decomposition search to sums of hotter games. The most important goals are to avoid the explicit summation of combinatorial games, and to find fast algorithms for both optimal and approximately optimal play. The algorithms are designed to play a sum of games where each subgame is simple enough such that a complete analysis is possible, and local properties such as incentives, means or temperatures can be computed quickly.

## 3.1 Exact Search for Optimal Play

In exact search, the minimax value of a sum game for a given first player is determined by a global $\alpha\beta$ search. The effect of using locally computed information for move ordering and for move pruning is studied and compared against standard search techniques.

**Move Ordering** In order to maximize the number of cutoffs in the search tree in $\alpha\beta$ search, good move ordering is essential. One of the most effective standard move ordering techniques is to use iterative deepening, and try the best move from the previous iteration first. This is a generic technique that does not use any game-specific features, or the structure of sum games.

A natural way of using information about subgames for move ordering is to compute the temperatures of subgames and order moves starting from the hottest subgame. In the experiments all combinations of these two move ordering techniques are compared.

**Move Pruning** Local analysis allows local move pruning by computing incentives and removing moves with dominated incentives. The tradeoff between the overhead of computing incentives and the gain from pruning dominated moves cannot easily be determined analytically. In this paper it is evaluated empirically.

## 3.2 Approximate Search

Approximate search methods provide a bridge between local-only analysis methods such as Hotstrat, which have limited accuracy, and exact global search, which may be prohibitively expensive. Approximate search requires a heuristic evaluation function to evaluate non-terminal leaf nodes in the global search.

**Heuristic Evaluation of Sum Game Positions** A heuristic evaluation function should approximate the minimax value of a given sum game with a given first player. For example, in the sum game **** the minimax score for left to play is ****, and the evaluation function should predict this score as accurately as possible.

Two types of heuristic evaluation functions were investigated:

1. **Static evaluation** The static evaluation uses a locally computed property, the mean of each subgame. The overall evaluation is the sum of the means of all subgames.
2. **Hotstrat+ rollout** A leaf node is evaluated by the final result of the game that was completed using the Hotstrat+ strategy for both players.

The two methods have very different characteristics. Static evaluation is fast but quite inaccurate. Hotstrat+ rollouts are usually more precise, but are much slower to compute, especially for long games, because games have to be played out.

Many variations on these basic evaluation functions are possible, and more research is necessary to find possible improvements. One such improvement could be an evaluation bias for the player to move. Let $t_{max}$ be the temperature of the hottest subgame.

If a sum game is an "enriched environment" [1], then the first player can achieve a minimax score that is $t_{max}/2$ larger than the sum of the means. However, preliminary experiments with using such a modified static evaluation function in the sum games described below were inconclusive.

**Resource-bounded $\boldsymbol{\alpha\beta}$ Search** In order to use a heuristic evaluation function, a control policy must decide when to stop searching and evaluate a position statically. Two such policies for were investigated. Both are independent of the specific choice of evaluation function.

1. **Depth-bounded $\boldsymbol{\alpha\beta}$ search** A simple way of reducing search cost is to limit the maximum search depth. With increasing depth, the heuristic evaluation is applied in closer-to-terminal positions, which should result in increased accuracy. In the limit, if the search depth is sufficient to reach the end of the game along each searched branch, the method computes the optimal result.
2. **Temperature-bounded $\boldsymbol{\alpha\beta}$ search** This method uses a variable depth search with a temperature bound $b$. Only hot positions $p$ with $t_{max}(p) > b$ are searched deeper. All positions with $t_{max}(p) \leq b$ are evaluated statically. For $b = 0$, this method is equivalent to a full search. If $b \geq t_{max}(g)$ for the starting position $g$, only a static evaluation of $g$ is done without any search.

## 4 Experiments

A series of experiments was performed for both exact and approximate search, using two parameterized test domains with sums of abstract combinatorial games. The tests vary the complexity of individual subgames as well as the number of subgames in a sum. 100 different randomly generated problem instances were used to generate each data point. For exact algorithms, the total runtime (in seconds) for solving all instances is given. For approximate algorithms, both the runtime and the aggregated loss against an optimal player are reported.

### 4.1 Test Domain for Sum Games

The following simple method, similar to the one in [4], is used to generate random instances from a restricted class of combinatorial games. Let $rnd(n)$ be a function generating a random integer uniformly distributed in the interval $[0, n-1]$. Then a random combinatorial game $g = rcg(k, n)$ of *level* $k > 0$ with *size parameter* $n > 0$ can be generated as follows:

1. Build a complete binary tree with $k$ levels below a root node. The first out-edge of each node corresponds to a Left move, the other to a move by Right.
2. Enumerate the $2^k$ leaf nodes from right to left, such that the right-most node is node 1 and the left-most node is node $2^k$.
3. Assign integers to all leaf nodes. Value $v_i$ is assigned to the $i$-th leaf node as follows: $v_1 = 0$, $v_{i+1} = v_i + rnd(n)$.

Two properties of this particular generator, that do not hold for all hot combinatorial games, are that values of leaf nodes are monotonically increasing from right to left, and that in each nonterminal position each player has exactly one move.

The 2-level random games used in [4] correspond to games $g = rcg(2, 50) + rnd(50)$ in this framework. The experiments in this paper use 2-level games generated by $rcg(2, 50)$ and 3-level games $rcg(3, 50)$. An example of a $rcg(2, 50)$ game is $114|66||49|0$ and an example of a $rcg(3, 50)$ game is $237|191||145|124|||97|57||32|0$. Given a subgame generator $G$, a random *sum game* with $s$ subgames is created simply as the sum of $s$ random subgames generated by $G$.

### 4.2 Preliminary Experiment: Combinatorial Summation

**Table 1.** Time (in seconds) for solving sum games by combinatorial summation. N/C = not completed, out of memory.

| Subgames | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 2-level | 0.543 | 9.90 | 148 | N/C |
| 3-level | 132 | N/C | - | - |

This preliminary experiment motivates the need to develop a practical algorithm for playing sums of hot games. It confirms the claim from Section 2.2 of this paper that the combinatorial summation of subgames is not a viable algorithm for playing sums of hot games. Table 1 summarizes the results. Even for very small sums of five 2-level or three 3-level games, the algorithm fails due to the quickly growing complexity of computing the canonical form of these sums, and runs out of memory after a few minutes of computation.

### 4.3 Experiment 1: Move Ordering

Experiments 1 and 2 use a global $\alpha\beta$ search to solve sum games exactly. The first experiment investigates the performance of two move ordering schemes. *Sort by temperature (TEMP)* sorts all moves according to their temperature. *Best previous first (BEST-PREV)* implements the standard technique of trying the best move from a previous iteration of iterative deepening search before any other move. A basic $\alpha\beta$ search without any move ordering is also included for reference.

Tables 2 and 3 show the performance for 2-level and 3-level games respectively. For very small sums, the overhead of move ordering is greater than the gain. For larger sums, as expected, both move ordering schemes either alone or in combination perform much better than no ordering. It is very interesting that TEMP alone outperforms both variants where BEST-PREV is used. Even the combination of both methods, which first orders moves by temperature, then moves the previously best move to the front, is slightly but consistently worse than TEMP alone. This may be very surprising to practitioners from other games, where the BEST-PREV enhancement is considered indispensable.

It provides a strong indication of how well temperature works as a measure of move urgency in hot games.

To further study the behavior of these two move ordering heuristics, detailed statistics were collected for a series of 100 2-level games with 12 subgames, using the fastest engine including incentive pruning as in Experiment 2. In 80.2% of all searched positions, both heuristics ordered the best move first. In 10.9% of positions, the first-ranked TEMP move was optimal but the first-ranked BEST-PREV move was not. The opposite occurred in 4.3% of all positions, where BEST-PREV had a right move but TEMP did not. Finally, in 4.6% of all positions both heuristics favored a suboptimal move. This result is consistent with the observed better performance of TEMP.

**Table 2.** Performance of move ordering for 2-level games. Best result in bold.

| | Subgames | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| No ordering | **0.172** | **0.254** | **0.339** | 0.803 | 2.77 | 13.6 | 85.9 | 1862 |
| BEST-PREV | 0.231 | 0.288 | 0.393 | **0.709** | 2.13 | 9.09 | 45.7 | 293 |
| **TEMP** | 0.204 | 0.260 | 0.448 | 0.766 | **1.90** | **6.81** | **29.5** | **131** |
| BEST-PREV and TEMP | 0.262 | 0.412 | 0.491 | 1.09 | 2.31 | 10.8 | 34.5 | 182 |

**Table 3.** Move ordering performance for 3-level games.

| | Subgames | | | | |
|---|---|---|---|---|---|
| Method | 2 | 3 | 4 | 5 | 6 |
| No ordering | **0.347** | 1.55 | 7.80 | 59.8 | 956 |
| BEST-PREV | 0.452 | 1.21 | 2.88 | 18.4 | 152 |
| **TEMP** | 0.398 | **1.01** | **1.85** | **11.9** | **105** |
| BEST-PREV and TEMP | 0.502 | 1.11 | 1.92 | 13.0 | 129 |

### 4.4 Experiment 2: Move Pruning

Experiment 2 demonstrates the effect of pruning using incentives of moves. Tables 4 and 5 show the performance for 2-level and 3-level games. For comparison, the first row contains the result obtained with TEMP but without using incentive pruning, from line 3 in Tables 2 and 3. In the remaining rows, incentive pruning (INC) is turned on, leading to a huge reduction in search time in all combinations. For all but the smallest instances, the version using TEMP move ordering and INC pruning is consistently the fastest.

**Table 4.** Incentive pruning for 2-level games.

| | Subgames | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| TEMP | **0.204** | **0.260** | 0.448 | 0.766 | 1.90 | 6.81 | 29.5 | 131 | 812 | N/A | N/A | N/A |
| **TEMP + INC** | 0.229 | 0.296 | **0.353** | **0.403** | **0.630** | **1.93** | **4.29** | **8.44** | **19.9** | **62.4** | **162** | **772** |
| PREV + INC | 0.230 | 0.293 | 0.446 | 0.563 | 0.819 | 2.36 | 5.23 | 11.4 | 26.7 | 88.5 | 221 | 1082 |
| TEMP + PREV + INC | 0.227 | 0.275 | 0.433 | 0.494 | 0.780 | 2.02 | 4.89 | 10.3 | 23.1 | 78.8 | 192 | 892 |

**Table 5.** Incentive pruning for 3-level games.

| | Subgames | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| TEMP | **0.398** | 1.01 | 1.85 | 11.9 | 105 | 1276 | N/A | N/A |
| **TEMP + INC** | 0.424 | **0.782** | **1.22** | **4.15** | **12.2** | **20.4** | **161** | **791** |
| PREV + INC | 0.431 | 0.823 | 1.31 | 5.95 | 15.8 | 29.4 | 218 | 1210 |
| TEMP + PREV + INC | 0.440 | 0.815 | 1.30 | 5.34 | 14.0 | 27.0 | 196 | 901 |

## 4.5 Experiment 3: Approximation Algorithms

Experiment 3 compares the game-playing strength of different *approximation* algorithms when pitted against a perfect opponent, namely the player using TEMP + INC that performed best in Experiment 2. The errors given in the tables are the total loss in points incurred by the tested algorithm against the optimal player in a series of 100 games. The smaller the error, the closer the algorithm is to playing perfectly.

This experiment measures only the quality of the players, not the time they need. See Experiment 4 in the next section for an investigation of tradeoffs between execution time and errors. The approximation algorithms compared in this experiment are:

1. **Hotstrat / Hotstrat+** compute temperatures / pseudo-temperatures of each subgame, and play in the hottest game as explained in Section 2.1.
2. **Thermostrat** uses a well-known sum-playing strategy based on thermographs [2].
3. **Depth-bounded $\alpha\beta$ search, static evaluation** uses a fixed-depth 3 ply search with static sum-of-means evaluation.
4. **Depth-bounded $\alpha\beta$ search,** *Hotstrat+* **rollouts** uses a fixed-depth 3 ply search but with rollouts as evaluation.
5. **Temperature-bounded $\alpha\beta$ search, static evaluation** The temperature bound for a search of position $p$ is set to $b = 0.8 \times t_{max}(p)$
6. **Temperature-bounded $\alpha\beta$ search,** *Hotstrat+* **rollouts** uses the same search control as the previous method with a rollout evaluation.

The following abbreviations are used in the tables:

**TEMP-$\alpha\beta$** temperature-bounded $\alpha\beta$ search.

**HR** *Hotstrat+* rollout evaluation.
**SE** Static evaluation by sum of means.

**Table 6.** Total loss over 100 2-level games against perfect opponent. Best result in bold. Best static method in italic. (1st) = first player uses method, (2nd) = second player uses method.

| | Number of Subgames | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| Hotstrat (1st) | 66 | 68 | 100 | 94 | 151 | 207 | 194 | 205 | 211 | 186 | 195 | 194 |
| Hotstrat (2nd) | 79 | 111 | 123 | 185 | 137 | 148 | 166 | 176 | 190 | 251 | 169 | 208 |
| *Hotstrat+ (1st)* | 68 | *65* | *93* | *84* | *136* | *180* | *158* | *171* | *199* | *176* | *178* | *184* |
| *Hotstrat+ (2nd)* | *68* | *82* | *88* | *147* | *115* | *118* | *126* | *138* | *158* | *215* | *147* | *188* |
| Thermostrat (1st) | *44* | 104 | 121 | 106 | 165 | 181 | 190 | 224 | 217 | 193 | 183 | 192 |
| Thermostrat (2nd) | 79 | 110 | 120 | 183 | 135 | 146 | 157 | 174 | 188 | 248 | 161 | 192 |
| 3-ply $\alpha\beta$, SE (1st) | 26 | 62 | 68 | 62 | 122 | 142 | 148 | 167 | 189 | 201 | 183 | 212 |
| 3-ply $\alpha\beta$, SE (2nd) | **0** | 69 | 42 | 103 | 102 | 110 | 130 | 134 | 143 | 178 | 202 | 189 |
| 3-ply $\alpha\beta$, HR (1st) | **0** | **0** | 8 | 21 | 22 | 36 | 47 | 55 | 70 | 89 | 120 | 99 |
| 3-ply $\alpha\beta$, HR (2nd) | **0** | **0** | **1** | 13 | 18 | 20 | 51 | **33** | 67 | 86 | 79 | 92 |
| TEMP-$\alpha\beta$, SE (1st) | 46 | 87 | 103 | 150 | 325 | 348 | 322 | 378 | 501 | 552 | 538 | 601 |
| TEMP-$\alpha\beta$, SE (2nd) | 33 | 33 | 93 | 164 | 202 | 212 | 288 | 368 | 432 | 512 | 542 | 621 |
| **TEMP-$\alpha\beta$, HR** (1st) | **0** | **0** | **5** | **17** | **19** | **25** | **45** | **43** | **53** | **61** | **78** | **81** |
| **TEMP-$\alpha\beta$, HR** (2nd) | **0** | 2 | 17 | **11** | **10** | **19** | **39** | 36 | **54** | **62** | **59** | **71** |

Among the static methods, *Hotstrat+* is consistently the best. The differences to plain Hotstrat and Thermostrat are rather small.

Overall, *Hotstrat+* rollouts are more precise than static evaluation for the same search depth. 3-ply search with static evaluation is better than static *Hotstrat+* up to about ten 2-level subgames, then it becomes worse. 3-ply search with *Hotstrat+* rollouts is always better than *Hotstrat+* without search. The two results for temperature-bounded search are very different. With static evaluation, this search method does not perform well. However, with rollouts it has the smallest total error in most tests. As the first player in a large number of 3-level subgames, 3-ply search with rollouts also does well.

This experiment should be considered as a first exploration of the potential of the different methods. However, the comparison is not fair since the running times of the methods are very different. Static methods are much faster than search-based methods, and search using static evaluation is much faster than search using rollouts. The choice of the search parameters depth and temperature bound in this experiment is also rather arbitrary. The next experiment compares tradeoffs between running time and error of the same set of algorithms.

## 4.6 Experiment 4: Time-Error Tradeoffs

Figure 1 plots error versus time used for the three static and four search-based algorithms. The test runs used sum games consisting of twelve 2-level games. The search

**Table 7.** Total loss over 100 3-level games.

| | Number of Subgames | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Method | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Hotstrat (1st) | 56 | 135 | 183 | 251 | 195 | 315 | 330 | 366 |
| Hotstrat (2nd) | 71 | 187 | 262 | 260 | 268 | 274 | 258 | 291 |
| *Hotstrat+ (1st)* | *55* | *111* | *152* | *232* | *182* | *281* | *295* | *331* |
| *Hotstrat+ (2nd)* | *67* | 162 | *218* | *229* | *232* | *212* | *215* | *248* |
| Thermostrat (1st) | 70 | 285 | 282 | 334 | 231 | 386 | 334 | 349 |
| Thermostrat (2nd) | 93 | *159* | 246 | 248 | 241 | 247 | 252 | 280 |
| 3-ply $\alpha\beta$, SE (1st) | 54 | 111 | 120 | 212 | 202 | 265 | 278 | 294 |
| 3-ply $\alpha\beta$, SE (2nd) | 4 | 89 | 135 | 168 | 181 | 221 | 234 | 262 |
| 3-ply $\alpha\beta$, HR (1st) | **0** | **7** | 31 | 88 | 93 | 142 | **160** | **148** |
| 3-ply $\alpha\beta$, HR (2nd) | **0** | 23 | 30 | **63** | 88 | 174 | 202 | 195 |
| TEMP-$\alpha\beta$, SE (1st) | 57 | 192 | 262 | 387 | 531 | 662 | 780 | 902 |
| TEMP-$\alpha\beta$, SE (2nd) | 26 | 121 | 202 | 248 | 431 | 601 | 735 | 886 |
| TEMP-$\alpha\beta$, HR (1st) | **0** | 8 | **30** | **85** | **72** | **132** | 189 | **148** |
| TEMP-$\alpha\beta$, HR (2nd) | **0** | **22** | **15** | 72 | **56** | **118** | **168** | **158** |

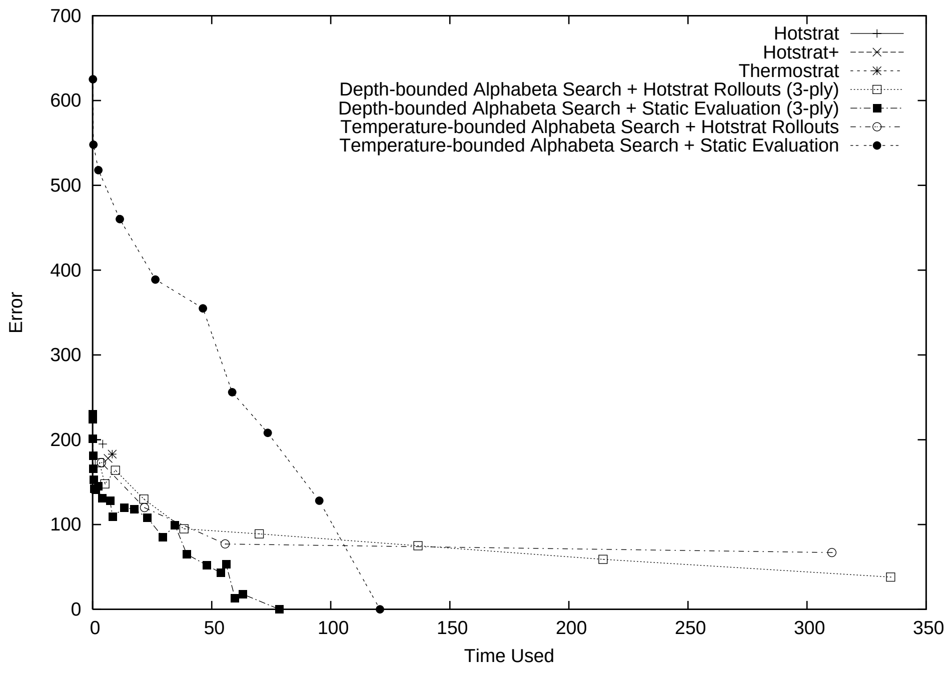

**Fig. 1.** Decrease of errors (points lost against optimal player) over time (in seconds) for the four search-based algorithms. For the three static methods, their single data point is shown.

depth of fixed-depth searches varies from 0 to 24 in increments of 1. Depth 24 represents a complete search, since each of the 12 subgames lasts at most 2 moves. For temperature-bounded search, the bound was set to $c \times t_{max}$, where $c$ varied from 0.0 to 1.0 in increments of 0.1.

The clear winner in these experiments is the simplest method: fixed-depth search with static evaluation. Even though the accuracy of rollouts is better for the same fixed search depth, as shown in Experiment 3 above, that evaluation is much too slow to be competitive. It remains unclear whether a different method can be found that achieves some of the precision of rollouts but is much faster. For example, depth-limited partial rollouts are an option.

Temperature-bounded search, which did very well combined with rollouts in the untimed experiment above, was also inferior in the timed experiment. The initial error of this method is very high. However, the slope of the error curve seems to be slanted more than for fixed-depth search, so there is hope that this method will perform well for more complex sums. More research is required.

## 5 Conclusions and Future Work

To the best of the authors' knowledge, this paper presents the first systematic experimental study of algorithms for efficiently solving sum games by search. For both exact and heuristic search in sum games, using local information in a global search is very effective. Move ordering by temperature works very well, and pruning of dominated incentives leads to a huge reduction in the search space. Both of these methods greatly improve a search-based sum game solver.

For heuristic search, a simple fixed-depth search with a sum of means static evaluation function performed best in the experiments. The two other techniques investigated, temperature-bounded search and *Hotstrat+* rollout evaluation, showed promise in untimed trials but were not competitive in their current form.

The main intended application of this method is for endgames in combinatorial games with many hot subgames, such as Amazons and Go. Based on the results with artificial games reported here, it seems feasible to evaluate such endgames with high accuracy. If the local games analysis itself is only approximate, as can be expected in complex endgame positions, the error of approximate sum game evaluation may be smaller than the local evaluation error in practice.

One important open question is whether real endgames behave similarly to the sums of abstract games explored in this paper. For example, the relative speed of position evaluation compared to the speed of sum game evaluation will certainly influence the results. As an intermediate step, before applying the method to full-scale Go or Amazons, a database of local endgame positions from such games could be built, and similar experiments repeated with sums of local positions from real games.

Finally, the question of a better random generator for hot combinatorial games should also be investigated. Maybe, specialized generators for more realistic "Go-like" or "Amazons-like" random games could be developed as well.

## Acknowledgements

The authors wish to thank NSERC (the Natural Sciences and Engineering Research Council of Canada) for financial support. Tristan Cazenave and Jean Mehat exchanged information about their implementation of classical sum game playing strategies.

## References

1. 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.
2. E. Berlekamp, J. Conway, and R. Guy. *Winning Ways*. Academic Press, London, 1982. Revised version published 2001-2004 by AK Peters.
3. E. Berlekamp and D. Wolfe. *Mathematical Go: Chilling Gets the Last Point*. A K Peters, Wellesley, 1994.
4. T. Cazenave. Comparative evaluation of strategies based on the values of direct threats. In *Board Games in Academia V*, Barcelona, 2002.
5. D.J. Moews. *On Some Combinatorial Games Connected with Go*. PhD thesis, University of California at Berkeley, 1993.
6. M. Müller. Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames. In *IJCAI-99*, pages 578–583, 1999.
7. M. Müller. Global and local game tree search. *Information Sciences*, 135(3–4):187–206, 2001.
8. W. Spight. Go thermography - the 4/21/98 Jiang-Rui endgame. In R. Nowakowski, editor, *More Games of No Chance*, pages 89–105. Cambridge University Press, 2002.
9. D. Wolfe. Go endgames are PSPACE-hard. In R. Nowakowski, editor, *More Games of No Chance*, pages 125–136. Cambridge University Press, 2002.
10. L. Yedwab. On playing well in a sum of games. Master's thesis, MIT, 1985. MIT/LCS/TR-348.