SELECTION SEARCH FOR MEAN AND TEMPERATURE OF MULTI-BRANCH COMBINATORIAL GAMES

Kuo-Yuan Kao[1], I-Chen Wu[2], Yi-Chang Shan[2] and Shi-Jim Yen[3]

Penghu, Taiwan[1], Hsinchu, Taiwan[2], Hualien, Taiwan[3]

ABSTRACT

This paper shows a new algorithm to calculate the mean and temperature of multi-branch combinatorial games. The algorithm expands gradually, one node at a time, the offspring of a game. After each step of expansion, the lower and upper bounds of the mean and temperature of the game are re-calculated. As the expanding process continues, the range between the lower and upper bounds is little by little narrowed. The key feature of the algorithm is its ability to generate a path of which the outcome is most likely to reduce the distance between the lower and upper bounds.

1. COMBINATORIAL GAMES

Combinatorial game theory studies two-player games with perfect information. The two players are assumed to take turns alternatively, and a game is considered as a sum of local positions, where each player can choose one local position to move at each turn. This section introduces a heap game, named *heap-go*, for illustrating some key ideas of combinatorial game theory.

Heap-go is played on a number of heaps of counters. Each counter has a weight and is coloured either blue or red. Figure 1 shows an example of heap-go setup. (Heap A and B are considered to be blue; heap E is red. Heap C and D are mixed; the top counter of C is red, the two other counters are blue.)

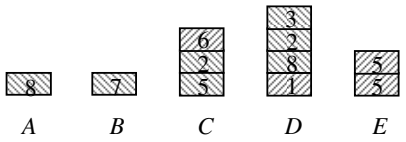


Figure 1: An example of heap-go setup.

Two players, *L* and *R*, move alternatively and their legal moves are different.

- When it is *L*'s turn to move, he[4] can choose any one of the heaps and repeatedly removes the top counter until either he removes a red counter or the heap has become empty.
- When it is *R*'s turn to move, he can choose any one of the heaps and repeatedly removes the top counter until either he removes a blue counter or the heap has become empty.

[1] Department of Information Management National Penghu University, Penghu, Taiwan. Email: stone@npu.edu.tw
[2] Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan. Email: icwu@csie.nctu.edu.tw, ycshan@java.csie.nctu.edu.tw.
[3] Department of Computer Science & Information Engineering, National Dong Hwa University, Hualien, Taiwan. Email: sjyen@mail.ndhu.edu.tw
[4] For brevity, we use 'he' and 'his' whenever 'he or she' and 'his or her' are meant.

The game is finished if all the counters in all the heaps are removed. The player who removed more total weights is the winner.

Heap-go is a two-player game with perfect information. The heaps are the local positions, where each player can choose one local position to move at each turn. The more heaps in a setup, the more options each player has. Although the complexity of the complete minimax game tree of a heap-go game may grow up exponentially with the increase of the total number of heaps, each local position can be represented as a simpler *combinatorial game tree* where each node represents the state of the local position, each left branch represents a L's move and each right branch a R's move at the local position.

Figure 2 shows the combinatorial game tree of heap C in Figure 1. The numbers at the terminal nodes are the net scores of the paths from the root to these nodes. L's scores are counted positive and R's negative. For example, consider the path LR. L gets 8 points for the first move (removed 2 counters); R gets 5 points for the second move (removed 1 counter); the net score is 3.

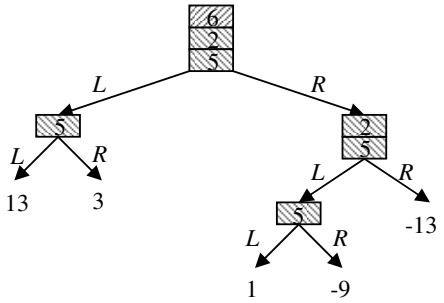


Figure 2: The game tree of heap C.

To follow the terminology of combinatorial game theory, each local position is called a *game*. If G is a game, then G^L (G^R) represents the set of L's (R's) options at the game, where each option in G^L (G^R) is a game after L's (R's) move at G. A game G is defined as an ordered pair of sets of games, and expressed as

$$G = \{G^L | \, G^R\} \tag{1}$$

When G is a terminal node in a game tree, it is represented by a numerical outcome value. For example, heaps A, B and C can be expressed as

$$A = \{8 \,|-8\},$$
$$B = \{7 \,|-7\},$$
$$C = \{\{13 \mid 3\} \mid \{\{1 \mid -9\} \mid -13\}\}.$$

Note that $A^L = 8$, $A^R = -8$, $B^L = 7$, $B^R = -7$, $C^L = \{13 \mid 3\}$, and $C^R = \{\{1|-9\}|-13\}$.

In combinatorial game theory, the sum of two games is a game. Let G and H be two games, the sum $G + H$ is defined as

$$G + H = \{G^L + H, G + H^L \mid G^R + H, G + H^R\}. \tag{2}$$

When G is a game and x is a number, the sum can be simplified as

$$G + x = \{G^L + x \mid G^R + x\} \tag{3}$$

$$x + G = \{ x + G^L \mid x + G^R \} \tag{4}$$

For example, the sum of $A = \{8 \mid -8\}$ and $B = \{7 \mid -7\}$ is

$$\begin{aligned} A + B &= \{A^L + B, A + B^L \mid A^R + B, A + B^R\} \\ &= \{8 + \{7 \mid -7\}, \{8 \mid -8\} + 7 \mid -8 + \{7 \mid -7\}, \{8 \mid -8\} - 7\} \\ &= \{\{15 \mid 1\}, \{15 \mid -1\} \mid \{-1 \mid -15\}, \{1 \mid -15\}\} \end{aligned}$$

Figure 3 shows the game trees of A, B and $A + B$.

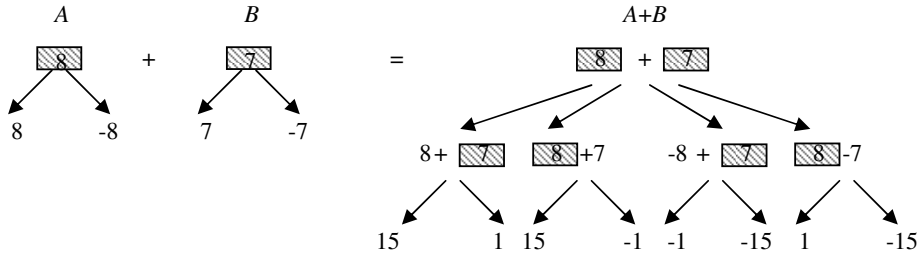


Figure 3: The game trees of A, B and $A + B$.

We use heap-go as the target game to test our new algorithm in this article. Two major reasons are as follows.

1. The maximum branching factor of a heap-go minimax tree can be controlled. A sum of m heaps has branching factor m in the minimax tree.
2. The maximum depth of a heap-go minimax tree can be controlled. A sum of heaps with a total of n counters has maximum depth n in the minimax tree.

For example, the complete minimax tree of a sum of the heaps in Figure 1 has branching factor 5 and maximum depth 11, since there are 5 heaps and 11 counters in total.

We end this section by two more observations at the heap-go game.

1. Consider the heaps A and E in Figure 1. Heap E is in favour of L, since L can get 10 points while R can only get 5 points if they make a move at heap E. In contrast, heap A is in favour of neither players, since both players will get the same score (8 points) if the move is at heap A. An important issue in combinatorial game theory is: how to measure the favours of a game, because the favours can help the estimation of the outcome of a game.
2. Consider the heaps A and B in Figure 1. There is only one counter in either heap A or heap B and the weight of the counter in heap A is heavier than the weight of the counter in heap B. Hence, both players will prefer a move at heap A over a move at heap B. In other words, the move size of heap A is larger than the move size of heap B. Both players will get 8 points if they move at heap A, and 7 points if they move at heap B. A second important issue in combinatorial game theory is: how to measure the move size of a game, because the move size can help the decision of choosing a good move.

2. MEAN AND TEMPERATURE

For each combinatorial game, there are two important values, *mean* and *temperature*. Roughly speaking, mean is a measure of the average outcome and temperature is a measure of the move size of a game. The existence of mean values of games was first raised and proved by Milnor (1953) and Hanner (1959). A

constructive algorithm, named *thermograph*, for mean and temperature was due to Berlekamp *et al.* (1982) and Conway (1976). An approach to calculating mean and temperature with partial information of a single branch game was proposed by Kao (1998). Müller *et al.* (2004) proposed to use a coupon stack CS with decreasing temperatures to simulate the environment and calculating the temperature of a game G by tracing the move sequence of the sum $G + CS$. Lew and Coulom (2010) proposed to estimate the mean and temperature of a game from its left and right stops and calculating these stops by temporal difference learning. In this paper, we continue our previous work and extend it to multiple-branch games.

In this paper, the mean and temperature of a game G is denoted as $m(G)$ and $t(G)$. A game G is called *hot* provided $t(G) > 0$.

Let G be a game and t be a number, define

$$L(G,t) = max_{\{x \in G^L\}} \begin{cases} m(G) \\ R(x,t) - t \end{cases} \tag{5}$$

$$R(G,t) = min_{\{y \in G^R\}} \begin{cases} m(G) \\ L(y,t) + t \end{cases} \tag{6}$$

$L(G,t)$ is called the *left wall* and $R(G,t)$ the *right wall* of G.

$L(G,t)$ ($R(G,t)$) is the min-max optimal outcome of the game G when L (R) moves first and subject to the constraint that L has the right either to accept the mean of G as the outcome or to make a move at G and pay a tax t. When G is a number (terminal position), both players will accept the number (equals its mean) as the outcome value and make no more move. Note that, for non-number games, the following five points hold.

1. When the tax t is low, the players may prefer to make a move and pay the tax t than accept the mean as the outcome.
2. When the tax t is too high, the players may prefer to accept the mean as the outcome than make a move and pay the tax t.
3. $L(G,t)$ is monotonically decreasing with respect to t. The higher the tax t, the lower the optimal outcome value when L moves fist.
4. $R(G,t)$ is monotonically increasing with respect to t. The higher the tax t, the higher the optimal outcome value when R moves fist.
5. When the tax t is low, we have $L(G,t) > m(G) > R(G,t)$. When the tax t reaches or exceeds some critical value, we have $L(G,t) = m(G) = R(G,t)$.

Thus, finding the mean and temperature of a game G is indeed a task of solving the min-max equation below.

$$max_{\{x \in G^L\}} R(x,t) - t = min_{\{y \in G^R\}} L(y,t) + t \tag{7}$$

There might be more than one solution of t for the above equation. The minimum solution of t is $t(G)$. When $t = t(G)$, the solution of the min-max equation equals $m(G)$.

Mean and temperature of a game have the following properties. Let G and H be two games. We have

$$m(G + H) = m(G) + m(H) \tag{8}$$

$$t(G + H) \leqq max\{t(G), t(H)\} \tag{9}$$

Hence, knowing the mean and temperature of games in a sum can help the estimation of the mean and temperature of the sum. Another important feature of mean and temperature is that they can be used to estimate the range of the optimal outcome of a game. The inequality below shows the bounds of the optimal outcomes $L(G,t)$ and $R(G,t)$.

$$m(G) - t(G) \leqq R(G,t) \leqq m(G) \leqq L(G,t) \leqq m(G) + t(G) \tag{10}$$

3. THERMOGRAPH

This section reviews the *thermograph* approach to calculating the mean and temperature of a game.
A function $f(t)$ is called *simple max* if it can be written as:

$$f(t) = max\{c_1, min\{c_2 - t, \ldots max\{c_{2k-1}, min\{c_{2k} - t, \ldots\}\}\}\} \tag{11}$$

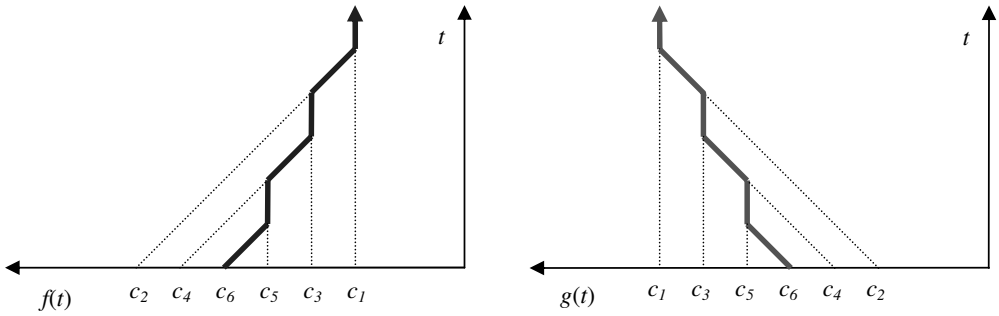
where $c_{2k} > c_{2k+2}, c_{2i} > c_{2k+1} > c_{2k-1}$.

Similarly, $g(t)$ is called *simple min* if it can be written as:

$$g(t) = min\{c_1, max\{c_2 + t, \ldots\ min\{c_{2k-1}, max\{c_{2k} + t, \ldots\}\}\}\} \tag{12}$$

where $c_{2k-1} > c_{2k+1} > c_{2i}, c_{2k+2} > c_{2k}$.

Each simple max (min) function can be represented as a sequence $[c_1, \ldots\ c_n]$ of constants. The graph of a simple max (min) function is a folded line. Figure 4 (a) and (b) show the graph of simple max and min functions by a black folded line and a grey folded line, respectively. Here black stands for blue, and grey stands for red. Note that the vertical axis labels t, the horizontal axis labels $f(t)$ and greater $f(t)$ value grows toward the left (instead of the right). The reason of this *unusual* convention (Berlekamp *et al.* (1982) and Conway (1976)) is to help the visualization of the advantage toward Left (L) or Right (R), since L prefers a greater value and R a smaller value.



(a) (b)

Figure 4: Thermographs of (a) a simple max function and (b) a simple min function.

It should be clear that the max (min) of two simple max (min) functions is again a simple max (min) function. If $f(t)$ is a simple max function and c is a number, then $min\{c, f(t) + t\}$ is a simple min function. If $g(t)$ is a simple min function and c is a number, then $max\{c, g(t) - t\}$ is a simple max function. Thus, the left wall of a game is a simple max function and the right wall is a simple min function. The thermograph of a game is a combined graph of the left and right walls. Figure 5 illustrates the thermograph of $G = \{3|\{0|-2\}\}$. The black (blue) line and grey (red) line coincide to a "black" (purple) line.

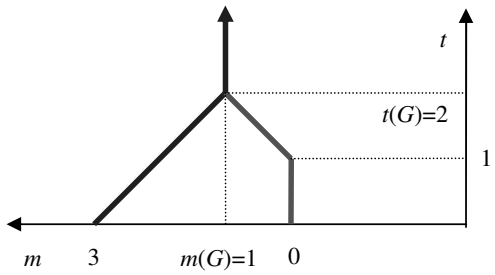


Figure 5: The thermograph of $G = \{3|\{0|-2\}\}$.

The general procedure to calculate the left wall (LW) and the right wall (RW) of a game $G$is as follows.

1. Calculate the walls of all G's children.
2. Find the max of the RWs of G's left children. Store the result as $R(t)$.
3. Find the min of the LWs of G's right children. Store the result as $L(t)$.
4. Calculate $m(G)$. That is to solve the equation $R(t) - t = L(t) + t$.
5. $LW = max\{m(G), R(t) - t\}$
6. $RW = min\{m(G), L(t) + t\}$

The above procedure is recursive. To calculate the walls of G, one needs to calculate the walls of all G's children first (Step 1). Eventually, the walls of all the offspring of G must be calculated in order to calculate the walls of G.

4. UPPER AND LOWER WALLS

In order to calculate the walls of a game, the thermograph approach requires visiting all the nodes of the game. In many applications, the number of a game's nodes could be a quite huge number, which makes the thermograph approach infeasible. However, partial information of a game's nodes could be used to estimate the lower and upper bounds of the game's walls (or mean and temperature), based on the procedure proposed by Kao (2000). For example, consider the game $H = \{\{x|20\}|4\}$. Although H^{LL}has the unknown value x, it can be shown $12 \leq m(H) \leq 20$ and $8 \leq t(H) \leq 16$, as long as $x \geq 20$. Sometimes, partial information of a game's nodes could be sufficient to determine its mean and temperature. For example, consider the game $= \{\{\{y|20\}|4\}|0\}$. Although G^{LLL}has the unknown value y, it can be shown $m(G) = 4$ and $t(G) = 4$, as long as $y \geq 20$.

For a hot combinatorial game G, the temperature is at least 0. It can be deduced that

$$m(G^L) \geq R(G^L, 0) = L(G, 0) \geq R(G, 0), \text{and} \tag{13}$$

$$m(G^R) \leq L(G^R, 0) = R(G, 0) \leq L(G, 0) \tag{14}$$

Equations (13) and (14) can be used to setup the lower or upper bound of a missing left or right child of a game. The assumptions $x \geq 20$ and $y \geq 20$ in the previous examples are reasonable provided the positions are hot games.

Let $m^U(G)$ and $m^L(G)$ denote the upper and lower bounds of $m(G)$, and $t^U(G)$ and $t^L(G)$ denote the upper and lower bounds of $t(G)$, respectively. We define the upper and lower bounds of the left and right walls of G as follows.

$$L^U(G,t) = max_{\{x \in G^L\}} \begin{cases} m^U(G) \\ R^U(x,t) - t \end{cases} \tag{15}$$

$$L^L(G,t) = max_{\{x \in G^L\}} \begin{cases} m^L(G) \\ R^L(x,t) - t \end{cases} \tag{16}$$

$$R^U(G,t) = min_{\{y \in G^R\}} \begin{cases} m^U(G) \\ L^U(y,t) + t \end{cases} \tag{17}$$

$$R^L(G,t) = min_{\{y \in G^R\}} \begin{cases} m^L(G) \\ L^L(y,t) + t \end{cases} \tag{18}$$

$m^U(G)$ can be derived by solving t from the equation,

$$max_{\{x \in G^L\}} R^U(x,t) - t = min_{\{y \in G^R\}} L^U(y,t) + t \tag{19}$$

and $m^L(G)$ can be derived by solving t from the equation,

$$max_{\{x \in G^L\}} R^L(x,t) - t = min_{\{y \in G^R\}} L^L(y,t) + t. \tag{20}$$

Similarly, $t^U(G)$ can be derived by solving t from the equation,

$$max_{\{x \in G^L\}} R^U(x,t) - t = min_{\{y \in G^R\}} L^L(y,t) + t, \tag{21}$$

and $t^L(G)$ can be derived by solving t from the equation,

$$max_{\{x \in G^L\}} R^L(x,t) - t = min_{\{y \in G^R\}} L^U(y,t) + t. \tag{22}$$

Figure 6 shows the relation between upper and lower walls and bounds of mean and temperature.

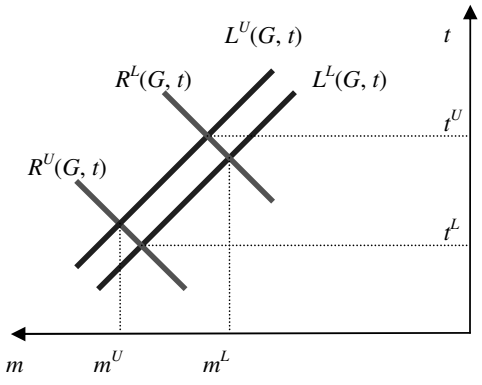


Figure 6: The relation between upper and lower walls and bounds of mean and temperature.

(Again black lines are blue, grey lines are red; henceforth this holds true.)

5. MT-SEARCH

This section presents a new algorithm, *MT-search*, to calculate the mean and temperature of games. The algorithm gradually, one node at a time, expands the offspring of a game. After each step of expanding, the lower and upper walls of all the nodes on the path from the new node to the root are re-calculated. As the expanding process continues, the distance between the lower and upper walls is narrowed. The algorithm terminates when either the distance between the upper and lower walls becomes 0 or the number of visited nodes reaches a given threshold, the maximum number of nodes to be visited. The choosing of an offspring node to expand is determined by some selection rules. These rules are introduced in later sections.

The MT-search algorithm is implemented in a game-independent engine named *MT-engine*. Application games can communicate with the engine through the MT-engine's interface procedures.

```
class MT_engine
{
  constructor MT-engine();
  destructor ~MT-engine();
  void explore(char path[]);
  void add-node(char path[], float value);
  float mean-UB();
  float mean-LB();
  float temp-UB();
  float temp-LB();
};
```

An application game can start an instance game by invoking the constructor of `MT-engine`. At the beginning, there is no visited node of the instance game. At each run, the application game calls the `explore()` procedure of `MT-engine` to get a path to be explored, and calls the `add_node()` procedure to add a node to the instance game of `MT-engine`. The `MT-engine` will update the upper and lower bounds of the mean and temperature each time after a new node has been added to the instance game. At any time, the procedures `mean_UB`, `mean_LB`, `temp_UB`, and `temp_UB` return the upper and lower bounds of the mean and temperature of the instance game.

The application game must provide a procedure `outcome(char path[])`, which returns the outcome of the specified path. The path is a string in the format $D_1n_1D_2n_2 \ldots D_kn_k \ldots D_mn_m$, where $D_k \in \{L, R\}$ is the direction of the child and n_k is an integer indicating the branching order of the child. For example *L3* indicates the 3rd left child of the root, *L3R2* indicates the 2nd right child of the 3rd left child of the root. If the specified path does not exist in the application, then `outcome()` returns a special value `NOT_EXISTS`. Otherwise, when the given path is not ending with a terminal node, the `outcome()` procedure automatically extends the path until it reaches a terminal node. The extended path must be in alternating directions and always selects the first child. For example, if the path *L1L2R3* is not a terminal node, then the path will be extended to *L1L2R3L1*, *L1L2R3L1R1*, or *L1L2R3L1R1L1* …, until it ends with a terminal node. Finally, the outcome procedure returns the expanded path.

Below is an example of feeding the MT-engine with 8 paths of a game with 10 or more nodes. The paths are generated by the engine. The application game inputs the values of the paths. With partial information of the game, the engine still can estimate the ranges of the mean and temperature of the game. After input 8 paths, the engine concludes the mean and temperature of the game. Figure 7 shows known paths of the game.

(1) L1R1L1=8
m = [-INF, 8.00] t = [0.00, INF]
(2) R1L1=-2
m = [-2.00, 8.00] t = [5.00, INF]
(3) R2L1=2
m = [-2.00, 8.00] t = [5.00, INF]
(4) R1R1L1=-10
m = [-1.00, 8.00] t = [7.00, INF]
(5) L2R1=-2
m = [-1.00, 8.00] t = [7.00, INF]
(6) L1L1=16
m = [-1.00, 3.00] t = [7.00, 11.00]
(7) R1R1R1=-16
m = [0.25, 2.25] t = [7.75, 10.00]
(8) L1R1R1=2
m = [1.50, 1.50] t = [9.00, 9.00]

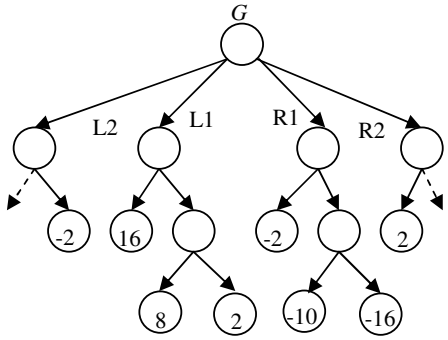


Figure 7: A game with incomplete information.

In the above calculation, there are two assumptions.

1. When the left (right) option of a game G is empty, the MT-Engine assumes there is a left (right) branch with values ranging between $R(G,0)$ and INF ($L(G,0)$ and –INF).
2. When there is at least one left (right) option, the MT-Engine assumes no more extra left options.

Note that when the MT-Engine explores more extra left options, the upper and lower bounds may be overridden.

6. UNCERTAINTY AND STABILITY

The MT-engine has no specific domain knowledge about the application games. The key feature of the MT-engine is its ability to generate a path whose outcome is most likely to reduce the distance between the lower and upper walls.

The first task of the MT-engine is to decide the direction, left or right, to explore. The decision is based on the uncertainties of the walls and the stability of the current node.

We define left and right uncertainty of G as:

$$L_{uncertainty}(G) = \sqrt{(m^{UL} - m^{LL})^2 + (t^{UL} - t^{LL})^2} + \sqrt{(m^{UU} - m^{LU})^2 + (t^{UU} - t^{LU})^2} \quad (23)$$

$$R_{uncertainty}(G) = \sqrt{(m^{UL} - m^{UU})^2 + (t^{UL} - t^{UU})^2} + \sqrt{(m^{LL} - m^{LU})^2 + (t^{LL} - t^{LU})^2} \quad (24)$$

where (m^{UU}, t^{UU}), (m^{LL}, t^{LL}), (m^{UL}, t^{UL}) and (m^{LU}, t^{LU}) are the solutions of equations (19) to (22).

Figure 8 and Figure 9 show the uncertainty values in a thermograph. $L_{uncertainty}$ ($R_{uncertainty}$) measures the distance between the lower and upper left (right) walls. The higher the uncertainty value is, the more likely a path in that direction will return useful information. Thus, the MT-engine tends to select the direction with greater uncertainty value. In Figure **8**, $L_{uncertainty}$ is less than $R_{uncertainty}$. The uncertainty value will reduce to 0 once the upper wall matches the lower wall. Note that the line segments in the walls are either vertical or with ± 1 slope as shown in Figure 9. Hence, the two square-root terms in (23) or (24) may not be equal.

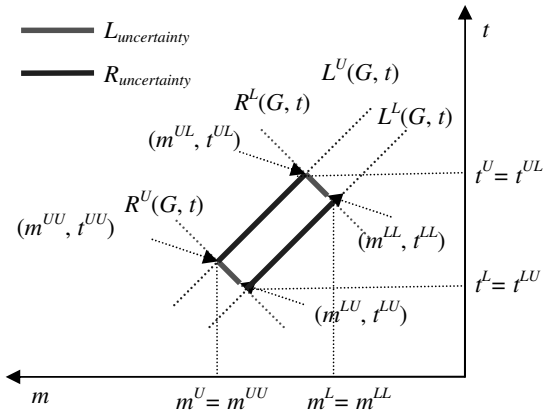


Figure 8: $L_{uncertainty}$ and $R_{uncertainty}$.

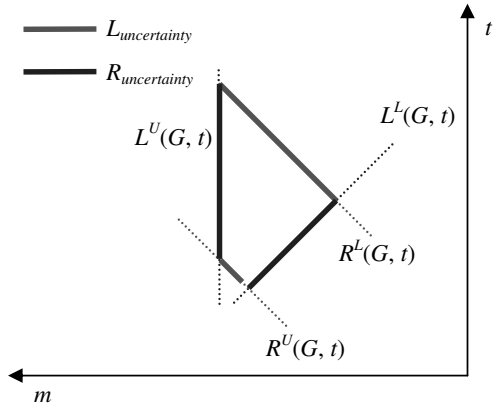


Figure 9: The line segments in the walls are either vertical or with ± 1 slope.

We say a game G is *stable* under t provided $t(G) < t$, and *unstable* provided $t(G) \geq t$. In our implementation, the value t is the minimum of the temperatures of G's ancestors. Consider game G as a right child of its parent as shown in Figure 10.

- When $t(G) < t, L(G,t) = m(G)$, the value of G^R will impact the value of $m(G)$, hence, it is still necessary to visit the nodes in G^R.
- When $t(G) \geq t, L(G,t) = max\ R(G^L, t) - t$, the value of G^R will not impact the left wall of G, and thus the right wall of G's parent. When this happens, there is no need to visit the nodes in G^R.

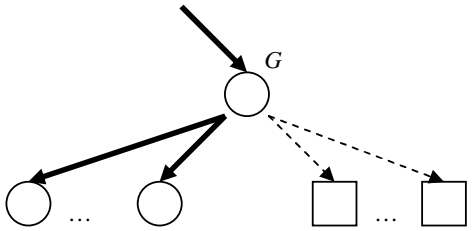


Figure 10: Exploring an unstable node.

Similarly, if G is an unstable left child of its parent, there is no need to visit the nodes in G^L.

In general, one may not be able to determine whether a node is stable or not during the search. We define the *stability* of G under t is defined as:

$$p_{stable}(G,t) = \begin{cases} 0, & if\ \ t^L(G) \geq t \\ 1, & if\ \ t^U(G) < t \\ (t - t^L(G))/(t^U(G) - t^L(G)) & otherwise \end{cases} \quad (25)$$

Stability and uncertainty can be combined together to determine the direction of exploration.
If G is a right child then explore its left children when

$$L_{uncertainty}(G) \geq R_{uncertainty}(G) \times p_{stable}(G,t) \quad (26)$$

Otherwise, explore right children.

If G is a left child then explore its right children when

$$R_{uncertainty}(G) \geq L_{uncertainty}(G) \times p_{stable}(G,t) \quad (27)$$

Otherwise, explore its left children.

Note that the values of uncertainty and stability depend on the bounds of $m(G)$ and $t(G)$. In real implementation, the bounds of $m(G)$ and $t(G)$ may not be accurate during the search. Especially, when either uncertainty or stability becomes 0, the search algorithm will block one direction to explore. To avoid this problem, one can introduce some noise to both sides of (26) and (27).

7. MINIMUM NUMBER OF PROVING NODES

MT-search can produce an estimation of the mean and temperature of a game even if there are only a few visited nodes. But how good is the estimation? In this section, we discuss the number of nodes required to produce a meaningful estimation.

Let G be a game with branching factor m (each player has m options at any non-terminal node) and depth n. The total number of nodes $T(m,n)$ of G is

$$T(m,n) = (2m)^n, m > 0, n > 0. \quad (28)$$

Define $f(m,n)$ as the least number of nodes required to prove the optimal left outcome, $L(G,0)$. Then

$$f(m,0) = 1,$$

$$f(m,1) = m,$$

$$f(m,n) = f(m,n-1) + (m-1) \times f(m,n-2), n > 1 \quad (29)$$

A node H is called *primary* if it is the best child of its parent node, otherwise it is called *secondary*. The least number of nodes occurs when the first branches are always the primary branches. The primary branch of a tree with branching factor m and depth n requires at least $f(m,n-1)$ nodes, all other branches require at least $f(m,n-2)$ nodes whether an α or a β cut is applied, as shown in Figure 11. When n approaches infinity, $f(m,n)$ approaches $\sqrt{m}^n$.

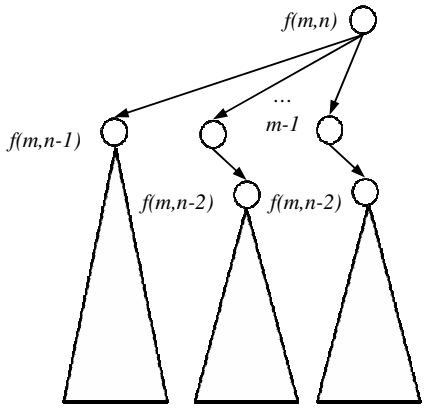


Figure 11: Scenario of deriving $f(m,n)$.

Define $s(m,n)$ as the least number of nodes required to prove G to be stable and determine the values of $m(G)$ and $t(G)$, $u(m,n)$ as the least number of nodes required to prove G to be unstable, $p(m,n)$ as the least number of nodes required to prove node G to be primary. For each primary node G, one either determines the value of $m(G)$ and $t(G)$ (when G is stable) or proves G unstable. Assume the odds for a primary node been stable and unstable is even, then

$$p(m,n) = [s(m,n) + u(m,n)]/2, n > 0. \quad (30)$$

To determine the mean and temperature of a stable node, one needs to visit all left and right branches. Both the left and right primary branches need at least $p(m,n-1)$ nodes, while all other secondary branches need at least $f(m,n-2)$ nodes due to an α or a β cut as shown in Figure 12.

$$s(m,0) = 1$$

$$s(m,1) = 2m$$

$$s(m,n) = 2 \times [p(m,n-1) + (m-1) \times f(m,n-2)], n > 1 \quad (31)$$

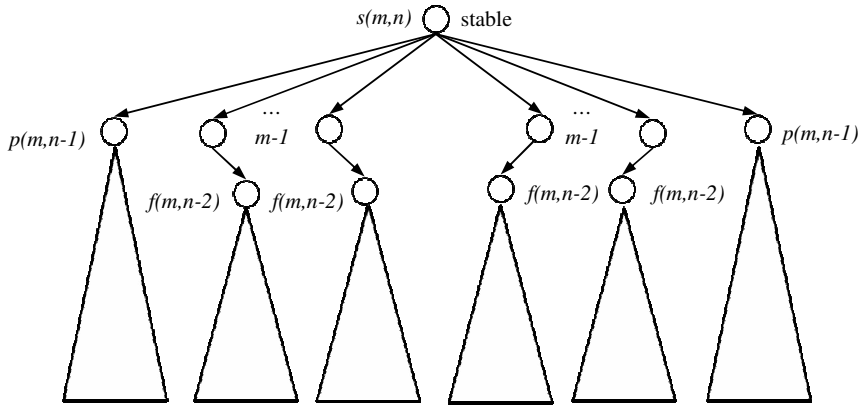


Figure 12: Scenario of deriving $s(m,n)$).

To prove an unstable node, one needs to show one of the right primary (or left) branches has a sufficient small (big) value and visit all the left (or right) branches among which one is primary and all others are secondary. The primary right (or left) branch requires at least $f(m,n-1)$ nodes; the primary left (or right) branch requires at least $p(m,n-1)$ nodes, the secondary branches require at least $f(m,n-2)$ nodes as shown in Figure 13.

$$u(m,0) = 1$$

$$u(m,1) = m+1$$

$$\begin{aligned} u(m,n) &= f(m,n-1) + (m-1) \times f(m,n-2) + p(m,n-1) \\ &= f(m,n) + p(m,n-1), n > 1 \end{aligned} \tag{32}$$

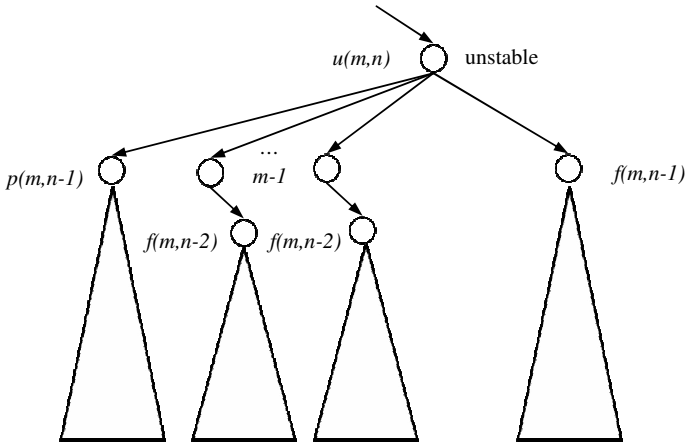


Figure 13: Scenario of deriving $u(m,n)$.

Table 1 shows the values of the ratios $c(m,n) = s(m,n)/\sqrt{T(m,n)}$ for selected ranges of m and n. When m is between 2 and 5, and n is less than 10, $s(m,n)$ is about of the same order of $\sqrt{T(m,n)}$. In general, $c(m,n)$ decreases as n increases. This result indicates that, to produce an accurate output of mean and temperature, any algorithm needs to visit at least $c\sqrt{T(m,n)}$,with $c > 1$, when m, n are in the ranges in Table 1.

$n\backslash m$	2	3	4	5
1	2.00	2.45	2.83	3.16
2	2.25	2.33	2.38	2.40
3	2.44	2.45	2.50	2.56
4	2.39	2.19	2.09	2.02
5	2.26	1.97	1.84	1.79
6	2.07	1.69	1.52	1.43
7	1.86	1.44	1.28	1.19
8	1.64	1.21	1.05	0.96
9	1.43	1.01	0.86	0.79
10	1.23	0.84	0.71	0.64

Table 1: The ratios $c(m, n)$.

8. EXPLORATION AND EXPLOITATION

Once one determined the direction to explore, the next step of the MT-engine is to select a child in this direction to explore. The general principle is selecting moves in the tree such that good moves are searched more often than moves that appear to be bad. One way of doing this is defining the weighted frequency as below, and selecting the child with minimum weighted frequency for exploration according to

```
if (b_primary(child))    child.freq = child.visits;
else                     child.freq=child.visits*K;
```

In our implementation, K depends on the *stability* of the *primary* child, the *branching factor* m and depth n of the *primary* child. When the primary child is stable,

$$K_s = \frac{s(m,n)}{f(m,n-1)}. \tag{33}$$

When the primary child is unstable,

$$K_u = \frac{u(m,n)}{f(m,n-1)}. \tag{34}$$

The general formula is

$$K = (K_s \times p_{stable}) + \big(K_u \times (1 - p_{stable})\big). \tag{35}$$

Table 2 and Table 3 show the values of K_s and K_u for a selected range of m and n. Note that in the first place for a given m and n, K_s is about 50% greater than K_u , and secondly that both K_s and K_u increase as the depth n increases.

In practice, the branching factor and depth of a game G may not be available. These parameters may be estimated by statistics of the visited nodes of G. The default value can be K_s =9 and K_u = 6.

$n\backslash m$	2	3	4	5
1	4.00	6.00	8.00	10.00
2	4.50	4.67	4.75	4.80
3	6.50	7.20	8.07	9.00
4	7.90	7.36	7.18	7.10
5	9.63	8.67	8.68	8.98
6	11.19	9.07	8.46	8.24
7	12.83	9.75	9.06	8.99
8	14.44	10.12	9.05	8.71
9	16.06	10.52	9.29	9.00
10	17.68	10.78	9.32	8.90

Table 2: $K_s(m,n)$.

$n\backslash m$	2	3	4	5
1	3.00	4.00	5.00	6.00
2	3.25	3.33	3.38	3.40
3	4.67	5.00	5.43	5.89
4	5.50	5.18	5.08	5.03
5	6.56	5.90	5.84	5.95
6	7.54	6.21	5.81	5.66
7	8.55	6.60	6.12	6.02
8	9.54	6.84	6.15	5.92
9	10.55	7.08	6.28	6.06
10	11.54	7.25	6.31	6.02

Table 3: $K_u(m,n)$.

9. EXPERIMENTAL RESULTS

We use heap-go as the target game to test the MT-search algorithm. The calculation of the mean and temperature of a single heap is not a difficult task, because there are only n distinct states in the game tree of a heap with n counters and there is only one option for each player at every non-terminal position of the game tree. An efficient algorithm is provided by Kao (2000) to calculate the mean and temperature of a single heap.

Our goal here is not to solve or analyze heap-go, but to use it as a sample space for testing the performance of the MT-search algorithm at *multi-branch* combinatorial games. There are at least three reasons to use heap-go as a sample space.

1. The maximum branching factor of a sample game can be controlled. A sum of m heaps has branching factor m.
2. The maximum depth of a sample game can be controlled. A sum of heaps with total n counters has maximum depth n.
3. The range of the temperature of a sample game can be controlled. A sum of heaps with counter weights ranging from 1 to w has a temperature range from 1 to $2w$.

Note that the MT-search algorithm is not aware of the split of a sample game into subgames.

Table 4 shows the classes of sums of heaps used as sample space in our experiments. The branching factor ranges from 2 to 5; the maximum depth ranges from 6 to 14; the temperatures range from 1 to 20. These ranges are chosen to resemble the branching factor, depth and temperature of endgame positions of 19x19 go. For each class, 1,000 sample games are randomly generated. The colours of the counters are uniformly distributed among blue and red. The weights of the counters are uniformly distributed among 1 to 10. The rightmost 3 columns in Table 4 are the average number of nodes (*T*), average temperature (*t_avg*) and standard derivation of the temperature (*t_stddev*) of the sample games. For the

overall sample games, the average number of nodes is 58,445, the average temperature is 11.52 with a standard derivation 2.61.

Class (heap sizes)						*T*(average number of node)	*t_avg*	*t_stddev*
A	3	3				198	11.29	3.00
	4	4				1,092	11.77	3.28
	5	5				6,215	12.02	3.48
	6	6				35,720	12.18	3.38
	7	7				205,590	12.23	3.53
B	2	2	2			1,158	11.00	2.24
	3	2	2			3,281	11.53	2.41
	3	3	2			10,166	11.99	2.50
	3	3	3			34,533	12.42	2.51
	4	3	3			103,745	12.63	2.59
C	2	2	1	1		3,936	10.49	2.18
	2	2	2	1		65,304	11.17	2.13
	3	3	2	1		167,055	12.02	2.47
D	2	1	1	1	1	13,440	9.85	1.96
	2	2	1	1	1	52,800	10.63	2.08
	2	2	2	1	1	230,880	11.12	1.98
Average						58,445	11.52	2.61

Table 4: Classes of sums of heaps.

Since the colours and weights of the counters in the sample games are uniformly distributed, the average mean of the sample games is close to 0. On the other hand, since the walls in the thermograph have slops +/- 1, the error of mean is about the same order as the error of temperature. Thus we only focus on the error of temperature of the games in our experiments.

There are two types of sample games: the first type has random branches, while the second type has ordered branches. When the branches are ordered, they are ordered by the heap temperatures. For each sample game, the MT-engine visits a set of predefined numbers of nodes. The set of predefined numbers of nodes are in the form:

$$2\sqrt{m} \times T^d \text{ .....,} \quad (36)$$

where m is the number of heaps in the sample game, T is the total nodes of the sample game and d ranges from 0.5 to 0.75. The parameters are chosen to guarantee the number of visited nodes no less than the minimum proving nodes as discussed in section 6.
The output temperature of MT-search is compared with the exact temperature of each game, and the square errors of all sample games in the same class are summed. Finally, the mean square error and the standard error are calculated for each class.

Table 5 shows the standard error for each class, where sample games have random branches. The results are summarized as follows.

- When d = 0.5, the number of visited nodes is about the order of the number of minimum proving nodes, the output of MT-search has a standard error of 0.93 in average.
- When the value of d increased by 0.05, the standard error is decreased by about 20 to 25%.

		d					
Class		0.50	0.55	0.60	0.65	0.70	0.75
A	33	1.08	0.88	0.56	0.43	0.30	0.08
	44	0.95	0.83	0.50	0.47	0.25	0.23
	55	0.81	0.74	0.42	0.25	0.18	0.10
	66	0.76	0.63	0.46	0.34	0.18	0.18
	77	0.68	0.64	0.22	0.12	0.11	0.07
B	222	1.05	0.67	0.46	0.39	0.16	0.14
	322	0.93	0.81	0.53	0.30	0.20	0.18
	332	0.99	0.69	0.47	0.26	0.25	0.07
	333	0.77	0.66	0.28	0.23	0.20	0.19
	433	0.89	0.70	0.45	0.24	0.19	0.13
C	2211	1.06	0.84	0.68	0.38	0.21	0.21
	2221	0.90	0.62	0.54	0.35	0.29	0.17
	3321	1.18	0.75	0.44	0.34	0.24	0.19
D	21111	1.04	0.58	0.32	0.33	0.20	0.08
	22111	0.97	0.66	0.49	0.31	0.26	0.08
	22211	0.78	0.59	0.42	0.27	0.22	0.17
Average		0.93	0.71	0.45	0.31	0.22	0.14

Table 5: Standard error for each class (random branches).

With sample space temperature ranging from 1 to 20, a standard error of less than 1.0 is a quite promising result.

Tables 6 shows the standard error for each class, where branches of sample games are ordered by temperature. The result indicates that branch ordering can significantly improve the accuracy (or efficiency) of the algorithm.

		d					
Class		0.50	0.55	0.60	0.65	0.70	0.75
A	33	0.92	0.55	0.32	0.15	0.05	0.01
	44	0.58	0.39	0.30	0.14	0.04	0.00
	55	0.45	0.26	0.15	0.09	0.02	0.00
	66	0.39	0.17	0.16	0.03	0.01	0.00
	77	0.24	0.12	0.03	0.01	0.00	0.00
B	222	0.58	0.47	0.20	0.09	0.02	0.00
	322	0.58	0.31	0.21	0.06	0.04	0.00
	332	0.43	0.29	0.15	0.08	0.01	0.00
	333	0.50	0.23	0.11	0.03	0.04	0.00
	433	0.35	0.20	0.07	0.07	0.02	0.00
C	2211	0.63	0.38	0.19	0.10	0.04	0.00
	2221	0.44	0.26	0.18	0.06	0.05	0.00
	3321	0.34	0.19	0.05	0.04	0.02	0.00
D	21111	0.48	0.29	0.24	0.16	0.09	0.00
	22111	0.31	0.24	0.16	0.09	0.06	0.00
	22211	0.30	0.20	0.11	0.07	0.04	0.00
Average		0.47	0.28	0.16	0.08	0.03	0.00

Table 6: Standard error for each class (ordered branches).

We summarize the key features of MT-search below.

(1) Branch ordering is critical in MT-search. Good move should be explored earlier before bad moves. Figure 14 shows comparison of the standard error between sorted and unsorted branches.

(2) The value of uncertainty and stability of a game can help the performance of MT-search. Figure 15 shows comparison of the standard error between MT-search with and without applying uncertainty and stability of a game to guide the search direction (left or right).

(3) A good balance between exploration and exploitation can help the performance of MT-search. Figure 16 shows comparison of the standard error between MT-search with and without applying the weighted frequency (cf. section 8).

In the experiments of Figure 14, Figure 15 and Figure 16, 1000 games are taken from the 3-3-2 class with an average number of nodes greater than 10,000. Compared with the nodes in the complete tree, the number of simulations (visited node) is relatively small. After 100 simulations (or less than 1% of the total nodes), MT-search already obtains a temperature with standard error less than 1.0.

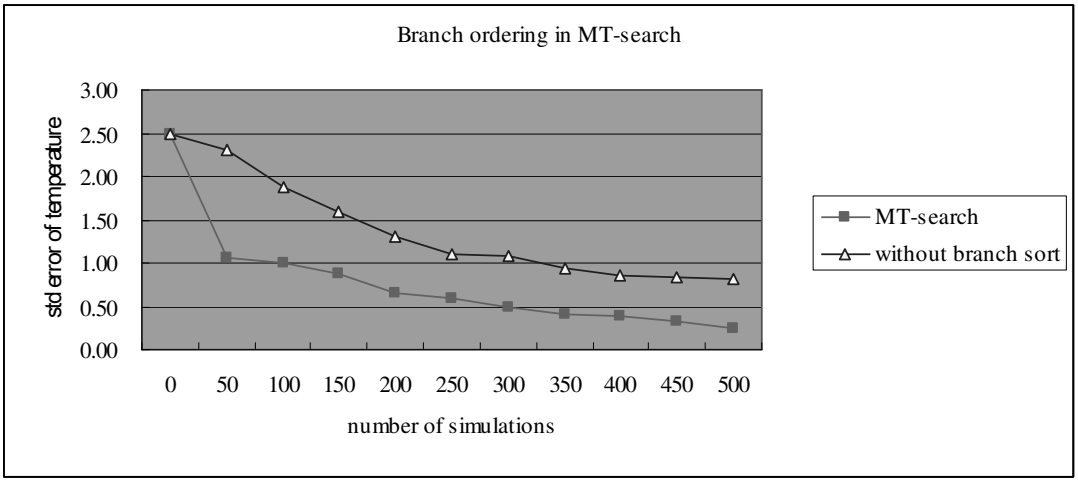


Figure 14: Branch ordering in MT-search.

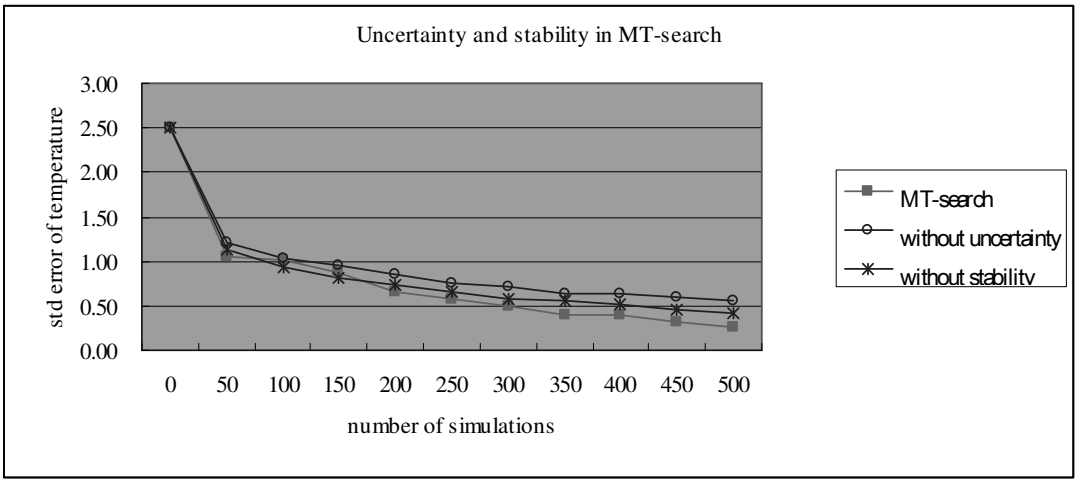


Figure 15: Uncertainty and stability in MT-search.

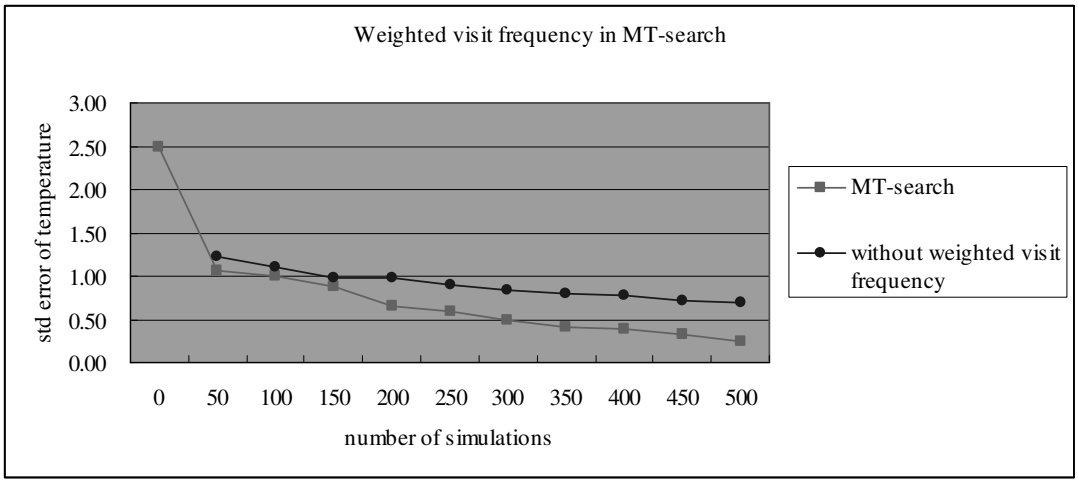


Figure 16: Weighted visit frequency in MT-search.

The weighted frequency scheme of MT-search follows the idea of Monte Carlo tree search (MCTS) by Coulom (2006), namely good moves are searched more often than bad moves. In MCTS, moves with greater expected action values are considered as better moves, while, in MT-search, moves (options) with greater expected walls (functions of t) are considered as better moves. Various MCTS schemes apply different formulas to determine the ratio of the frequencies between exploration of new possibilities and the exploitation of old certainties. In this paper is that the ratio is derived from the estimated minimum numbers of proving nodes for the primary and the secondary moves (see section 8). Although Figure 16 shows significant improvement, compared with random paths, has been obtained by applying the weighted frequency scheme of the minimum numbers of proving nodes, other weighted frequency schemes are still open for further research.

10. CONCLUSIONS

Since the 1970s, combinatorial game theory has become the common fundamental mathematical model for the analysis of many intelligent games. Mean and temperature are the most important concepts for hot combinatorial games.

In this paper, we presented an efficient algorithm to calculate the mean and temperature of multi-branch hot games. Moreover, we implemented the search algorithm in a game-independent search engine. The search engine has a straightforward interface; computer game programs can apply the engine easily. The key feature of the MT-engine is its ability to generate a path of which the outcome is most likely to reduce the distance between the lower and upper walls.

MT-search can output high quality outcomes by searching a portion of the game tree. Given a game G with T nodes, the algorithm can output (1) an answer with standard error less than 5% of the range of the sample space temperature, after visiting $4 \times \sqrt{T}$ nodes. Our experimental results also indicate (2) the importance of the branch ordering, (3) the importance of uncertainty and stability of games, and (4) the importance of weighted visit frequency in MT-search.

ACKNOWLEDGEMENTS

The authors would like to thank the National Science Council of the Republic of China (Taiwan) for financial support of this research under contract number NSC 99-2221-E-009-102-MY3 and NSC 100-2221-E-346-007.

11. REFERENCES

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways for your Mathematical Plays*, Academic Press, New York, Chapter 6.

Conway, J. H. (1976). *On Numbers and Games*, Academic Press, New York, Chapter 9.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search, *5th International Conference on Computer and Games*, Volume 4630 of Lecture Notes in Computer Science (LNCS), pages 72-83. Springer, Heidelberg, Germany.

Hanner, O. (1959). Mean Play for Sums of Positional Games, *Pacific J. Math*. 9, pp. 81-99.

Kao, K.-Y. (1998). Mean and Temperature Search for Combinatorial Games, *JCIS Proceedings*, Vol. I, pp. 389-392.

Kao, K.-Y. (2000). Mean and Temperature Search for Go Endgames, *Information Science* 122, pp. 77-90.

Kao, K-Y, Wu, I-C, and Shan Y-C. (2012). XT Domineering: A New Combinatorial Game, to appear in *Knowledge-Based Systems*.

Milnor, J. (1953). Sums of Positional Games, in Kuhn and Tucker (eds.) *Contributions to the Theory of Games,* Ann. Math. Studies #28, Princeton, pp. 291-301.

Lew, L., and Coulom, R. (2010). Simulation-based Search of Combinatorial Games, in *ICML Workshop on Machine Learning and Games*, Israel.

Müller, M., Enzenberger, M., and Schaeffer, J., (2004). Temperature discovery search, In *AAAI Proceedings*, pp. 658–663, San Jose, CA.

Silver, D. (2009). *Reinforcement Learning and Simulation-Based Search in Computer Go*. Doctoral dissertation, Department of Computing Science, University of Alberta, Canada.