Theory of Computer Games: Selected Advanced Topics

Tsan-sheng Hsu

徐讚昇

tshsu@iis.sinica.edu.tw

http://www.iis.sinica.edu.tw/~tshsu
Abstract

- Some advanced research issues.
  - The graph history interaction (GHI) problem.
  - Opponent models.
  - Searching chance nodes.
  - Proof-number search.
The graph history interaction (GHI) problem [Campbell 1985]:
- In a game graph, a position can be visited by more than one paths.
- The value of the position depends on the path visiting it.
  - It can be win, loss or draw for Chinese chess.
  - It can only be draw for Western chess.
  - It can only be loss for Go.

In the transposition table, you record the value of a position, but not the path leading to it.
- Values computed from rules on repetition cannot be used later on.
- It takes a huge amount of storage to store all the paths visiting it.

This is a very difficult problem to be solved in real time [Wu et al. ’05].
• Assume the one causes loops loses the game.

• \( A \rightarrow B \rightarrow E \rightarrow I \rightarrow J \rightarrow H \rightarrow E \) is loss because of rules of repetition.
  ▶ Memorized \( H \) as a loss position.

• \( A \rightarrow B \rightarrow D \) is a loss.

• \( A \rightarrow C \rightarrow F \rightarrow H \) is loss because \( H \) is recorded as loss.

• \( A \) is loss because both branches lead to loss.

• However, \( A \rightarrow C \rightarrow F \rightarrow H \rightarrow E \rightarrow G \) is a win.
Opponent models

- In a normal alpha-beta search, it is assumed that you and the opponent use the same strategy.
  - What is good to you is bad to the opponent and vice versa!
  - Hence we can reduce a minimax search to a NegaMax search.
  - This is normally true when the game ends, but may not be true in the middle of the game.

- What will happen when there are two strategies or evaluating functions $f_1$ and $f_2$ so that
  - for some positions $p$, $f_1(p)$ is better than $f_2(p)$
    - “better” means closer to the real value $f(p)$
  - for some positions $q$, $f_2(q)$ is better than $f_1(q)$

- If you are using $f_1$ and you know your opponent is using $f_2$, what can be done to take advantage of this information.
  - This is called OM (opponent model) search [Carmel and Markovitch 1996].
    - In a MAX node, use $f_1$.
    - In a MIN node, use $f_2$. 
Opponent models – comments

- **Comments:**
  - Need to know your opponent’s model precisely or to have some knowledge about your opponent.
  - How to learn the opponent model on-line or off-line?
  - When there are more than 2 possible opponent strategies, use a probability model (PrOM search) to form a strategy.
Search with chance nodes

- **Chinese dark chess**
  - Two player, zero sum, complete information
  - Perfect information
  - Stochastic
  - There is a chance node during searching [Ballard 1983].
    - The value of a node is a distribution, not a fixed value.

- **Previous work**
  - Alpha-beta based [Ballard 1983]
  - Monte-Carlo based [Lancoto et al 2013]
It’s black turn and black has 6 different possible legal moves including 4 of them being moving its elephant and two flipping moves at a1 or a8.
- It is difficult for black to secure a win by moving its elephant.
Example (2/3)

- If black flips a1, then it becomes one of the 2 followings cases.
  - If a1 is black cannon, then black may win.
  - If a1 is black king, then it is difficult for black to win.
Example (3/3)

- If black flips a8, then it becomes one of the 2 followings cases.
  - If a8 is black cannon, then it is difficult for black to win.
  - If a8 is black king, then black may lose.
Basic ideas for searching chance nodes

- Assume a chance node $x$ has a score probability distribution function $Pr(*)$ with the range of possible outcomes from 1 to $N$ where $N$ is a positive integer.
  - For each possible outcome $i$, we need to compute $score(i)$.
  - The expected value $E = \sum_{i=1}^{N} score(i) \times Pr(x = i)$.
  - The minimum value is $m = \min_{i=1}^{N} \{score(i) \mid Pr(x = i) > 0\}$.
  - The maximum value is $M = \max_{i=1}^{N} \{score(i) \mid Pr(x = i) > 0\}$.

- Example: open game in Chinese dark chess.
  - For the first ply, $N = 14 \times 32$.
    - Using symmetry, we can reduce it to 7*8.
  - We now consider the chance node of flipping the piece at the cell a1.
    - $N = 14$.
    - Assume $x = 1$ means a black King is revealed and $x = 8$ means a red King is revealed.
    - Then $score(1) = score(8)$ since the first player owns the revealed king no matter its color is.
    - $Pr(x = 1) = Pr(x = 8) = 1/14$. 

TCG: Selected advanced topics, 20170109, Tsan-sheng Hsuc © 2017
Bounds in a chance node

Assume the various possibilities of a chance node is evaluated one by one in the order that at the end of phase $i$, $i = N$ is evaluated.

- Assume $v_{min} \leq score(i) \leq v_{max}$.

How do the lower and upper bounds, namely $m_i$ and $M_i$, of the chance node change at the end of phase $i$?

- $i = 0$.
  - $m_0 = v_{min}$
  - $M_0 = v_{max}$

- $i = 1$, we first compute $score(1)$, and then know
  - $m_1 \geq score(1) \times Pr(x = 1) + v_{min} \times (1 - Pr(x = 1))$, and
  - $M_1 \leq score(1) \times Pr(x = 1) + v_{max} \times (1 - Pr(x = 1))$.

- ...

- $i = i^*$, we have computed $score(1), \ldots, score(i^*)$, and then know
  - $m_{i^*} \geq \sum_{i=1}^{i^*} score(i) \times Pr(x = i) + v_{min} \times (1 - \sum_{i=1}^{i^*} Pr(x = i))$, and
  - $M_{i^*} \leq \sum_{i=1}^{i^*} score(i) \times Pr(x = i) + v_{max} \times (1 - \sum_{i=1}^{i^*} Pr(x = i))$. 

TCG: Selected advanced topics, 20170109, Tsan-sheng Hsu ©
Algorithm: Chance_Search

- **Algorithm** $F^{2.1'}(\text{position } p, \text{ value } \alpha, \text{ value } \beta)$
  // max node
  - determine the successor positions $p_1, \ldots, p_b$
  - if $b = 0$, then return $f(p)$
    else begin
      ▶ $m := \alpha$
      ▶ for $i := 1$ to $b$ do
        ▶ begin
          ▶ if $p_i$ is to play a chance node $n$
            then $t := Star1\_F^{2.1'}(p_i, n, \alpha, \beta)$
          ▶ else $t := G^{2.1'}(p_i, m, \beta)$
          ▶ if $t > m$ then $m := t$
          ▶ if $m \geq \beta$ then return($m$) // beta cut off
        ▶ end
      ▶ end;
    ▶ return $m$
Algorithm: Chance_Search

- **Algorithm Star1_F2.1'** (position \( p \), node \( n \), value \( \alpha \), value \( \beta \))
  - // return the expected value of a chance node \( n \)
  - determine the possible values of the chance node \( n \) to be \( k_1, \ldots, k_c \)
  - \( m_0 = \alpha; \) // current lower bound, \( \alpha \geq v_{\min} \)
  - \( M_0 = \beta; \) // current upper bound, \( \beta \leq v_{\max} \)
  - \( v_{\text{sum}} = 0; \) // current expected value
  - for \( i = 1 \) to \( c \) do
    - begin
      - let \( p_i \) be the position of assigning \( k_i \) to \( n \) in \( p \);
      - \( t := G2.1'(p_i, \max\{m_{i-1}, v_{\min}\}, \min\{M_{i-1}, v_{\max}\}); \)
      - if \( t \leq m_{i-1} \) then \( t := \alpha; \)
      - if \( t \geq M_{i-1} \) then \( t := \beta; \)
      - \( v_{\text{sum}} += t \times Pr_i \)
      - \( m_i = m_{i-1} + (t - \alpha) \times Pr_i; \)
      - \( M_i = M_{i-1} + (t - \beta) \times Pr_i; \)
      - \( \ldots \)
    - end
  - return \( v_{\text{sum}}; \)
Example: Chinese dark chess

Assumption:

- The range of the scores of Chinese dark chess is $[-10, 10]$ inclusive, $\alpha = -10$ and $\beta = 10$.
- $N = 7$.
- $Pr(x = i) = 1/N = 1/7$.

Calculation:

- $i = 0$,
  - $m_0 = -10$.
  - $M_0 = 10$.
- $i = 1$ and if $score(1) = -2$, then
  - $m_1 = -2 \times 1/7 + -10 \times 6/7 = -62/7 \approx -8.86$.
  - $M_1 = -2 \times 1/7 + 10 \times 6/7 = 58/7 \approx 8.26$.
- $i = 1$ and if $score(1) = 3$, then
  - $m_1 = 3 \times 1/7 + -10 \times 6/7 = -57/7 \approx -8.14$.
  - $M_1 = 3 \times 1/7 + 10 \times 6/7 = 63/7 = 9$. 

TCG: Selected advanced topics, 20170109, Tsan-sheng Hsu ©
Comments

- We illustrate the ideas using a fail hard version of the alpha-beta algorithm.
  - Fail hard version has a simple logic in maintaining the search interval.
  - The semantic of comparing an exact returning value with an expected returning value is something that needs careful thinking.
  - May want to pick a chance node with a lower value but having a hope of winning not one with a slightly higher value but having no hope of winning when you are in disadvantageous positions.
  - May want to pick a chance node with a lower value but having no chance of losing, not one with a slightly higher value but having a chance of losing when you are in advantage positions.

- Need to revise algorithms carefully when dealing with the fail sort version or the NegaScout version.
  - What does it mean to combine bounds from a fail soft version?

- Exist other improvements by considering better move orderings involving chance nodes.
How to use these bounds

- The lower and upper bounds of the expected score can be used to do alpha-beta pruning.
  - Nicely fit into the alpha-beta search algorithm.
- Can do better by not searching the DFS order.
  - It is not necessary to search completely the subtree of $x = 1$ first, and then start to look at the subtree of $x = 2$.
  - Assume it is a MAX chance node, e.g., the opponent takes a flip.
    - Knowing some value $v'_1$ of a subtree for $x = 1$ gives an upper bound, i.e., $\text{score}(1) \geq v'_1$.
    - Knowing some value $v'_2$ of a subtree for $x = 2$ gives another upper bound, i.e., $\text{score}(2) \geq v'_2$.
    - These bounds can be used to make the search window further narrower.
- For Monte-Carlo based algorithm, we need to use a sparse sampling algorithm to efficiently estimate the expected value of a chance node [Kearn et al 2002].
Ideas for new search methods

- Consider the case of a 2-player game tree with either 0 or 1 on the leaves.
  - win, or not win which is lose or draw;
  - lose, or not lose which is win or draw;
  - Call this a binary valued game tree.

- If the game tree is known as well as the values of some leaves are known, can you make use of this information to search this game tree faster?
  - The value of the root is either 0 or 1.
  - If a branch of the root returns 1, then we know for sure the value of the root is 1.
  - The value of the root is 0 only when all branches of the root returns 0.
  - An AND-OR game tree search.
Which node to search next?

- **A most proving node** for a node $u$: a node if its value is 1, then the value of $u$ is 1.
- **A most disproving node** for a node $u$: a node if its value is 0, then the value of $u$ is 0.

\[
\begin{array}{c}
\text{d} & \text{e} & \text{f} & \text{g} & \text{h} \\
1 & ? & ? & 1 & ?
\end{array}
\quad
\begin{array}{c}
\text{d} & \text{e} & \text{f} & \text{g} & \text{h} \\
1 & ? & ? & 1 & ?
\end{array}
\]
Proof or Disproof Number

- Assign a **proof number** and a **disproof number** to each node $u$ in a binary valued game tree.
  - $\text{proof}(u)$: the minimum number of leaves needed to visited in order for the value of $u$ to be 1.
  - $\text{disproof}(u)$: the minimum number of leaves needed to visited in order for the value of $u$ to be 0.
Proof Number: Definition

- **u is a leaf:**
  - If \( \text{value}(u) \) is unknown, then \( \text{proof}(u) \) is the cost of evaluating \( u \).
  - If \( \text{value}(u) \) is 1, then \( \text{proof}(u) = 0 \).
  - If \( \text{value}(u) \) is 0, then \( \text{proof}(u) = \infty \).

- **u is an internal node with all of the children \( u_1, \ldots, u_b \):**
  - if \( u \) is a MAX node,
    \[
    \text{proof}(u) = \min_{i=1}^{i=b} \text{proof}(u_i);
    \]
  - if \( u \) is a MIN node,
    \[
    \text{proof}(u) = \sum_{i=1}^{i=b} \text{proof}(u_i).
    \]
Disproof Number: Definition

- $u$ is a leaf:
  - If $\text{value}(u)$ is unknown, then $\text{disproof}(u)$ is cost of evaluating $u$.
  - If $\text{value}(u)$ is 1, then $\text{disproof}(u) = \infty$.
  - If $\text{value}(u)$ is 0, then $\text{disproof}(u) = 0$.

- $u$ is an internal node with all of the children $u_1, \ldots, u_b$:
  - if $u$ is a MAX node,
    \[
    \text{disproof}(u) = \sum_{i=1}^{i=b} \text{disproof}(u_i);
    \]
  - if $u$ is a MIN node,
    \[
    \text{disproof}(u) = \min_{i=1}^{i=b} \text{disproof}(u_i).
    \]
Illustrations

proof number, disproof number
How to use these Numbers

- If the numbers are known in advance, then from the root, we search a child \( u \) with the value equals to \( \min\{\text{proof}(\text{root}), \text{disproof}(\text{root})\} \).
  - Then we find a path from the root towards a leaf recursively as follows,
    - if we try to prove it, then pick a child with the least proof number for a MAX node, and pick any node that has a chance to be proved for a MIN node.
    - if we try to disprove it, then pick a child with the least disproof number for a MIN node, and pick any node that has a chance to be disproved for a MAX node.

- Assume each leaf takes a lot of time to evaluate.
  - For example, the game tree represents an open game tree or an endgame tree.
  - Depends on the results we have so far, pick the next leaf to prove or disprove.

- Need to be able to update these numbers on the fly.
**PN-search: algorithm**

- **loop:** Compute or update proof and disproof numbers for each node in a bottom up fashion.
  - If $\text{proof}(\text{root}) = 0$ or $\text{disproof}(\text{root}) = 0$, then we are done, otherwise
    - $\text{proof}(\text{root}) \leq \text{disproof}(\text{root})$: we try to prove it.
    - $\text{proof}(\text{root}) > \text{disproof}(\text{root})$: we try to disprove it.

- $u \leftarrow \text{root}; \{ \ast \text{ find the leaf to prove or disprove } \ast \}$
  - if we try to prove, then
    - while $u$ is not a leaf do
      - if $u$ is a MAX node, then
        - $u \leftarrow$ leftmost child of $u$ with the smallest non-zero proof number;
      - if current is a MIN node, then
        - $u \leftarrow$ leftmost child of $u$ with a non-zero proof number;
  - if we try to disprove, then
    - while $u$ is not a leaf do
      - if $u$ is a MAX node, then
        - $u \leftarrow$ leftmost child of $u$ with a non-zero disproof number;
      - if current is a MIN node, then
        - $u \leftarrow$ leftmost child of $u$ with the smallest non-zero disproof number;

- Prove or disprove $u$; go to loop;
Multi-Valued game Tree

- The values of the leaves may not be binary.
  - Assume the values are non-negative integers.
  - Note: it can be in any finite countable domain.

- Revision of the proof and disproof numbers.
  - $\text{proof}_v(u)$: the minimum number of leaves needed to visited in order for the value of $u$ to $\geq v$.
    - $\text{proof}(u) \equiv \text{proof}_1(u)$.
  - $\text{disproof}_v(u)$: the minimum number of leaves needed to visited in order for the value of $u$ to $< v$.
    - $\text{disproof}(u) \equiv \text{disproof}_1(u)$. 
Illustration
Multi-Valued Proof Number

- \( u \) is a leaf:
  - If \( \text{value}(u) \) is unknown, then \( \text{proof}_v(u) \) is cost of evaluating \( u \).
  - If \( \text{value}(u) \geq v \), then \( \text{proof}_v(u) = 0 \).
  - If \( \text{value}(u) < v \), then \( \text{proof}_v(u) = \infty \).

- \( u \) is an internal node with all of the children \( u_1, \ldots, u_b \):
  - if \( u \) is a MAX node,
    \[
    \text{proof}_v(u) = \min_{i=1}^{i=b} \text{proof}_v(u_i);
    \]

  - if \( u \) is a MIN node,
    \[
    \text{proof}_v(u) = \sum_{i=1}^{i=b} \text{proof}_v(u_i).
    \]
Multi-valued Disproof Number

- **u is a leaf:**
  - If \( \text{value}(u) \) is unknown, then \( \text{disproof}_v(u) \) is cost of evaluating \( u \).
  - If \( \text{value}(u) \geq v \), then \( \text{disproof}_v(u) = \infty \).
  - If \( \text{value}(u) < v \), then \( \text{disproof}_v(u) = 0 \).

- **u is an internal node with all of the children \( u_1, \ldots, u_b \):**
  - if \( u \) is a MAX node,
    \[
    \text{disproof}_v(u) = \sum_{i=1}^{i=b} \text{disproof}_v(u_i);
    \]
  - if \( u \) is a MIN node,
    \[
    \text{disproof}_v(u) = \min_{i=1}^{i=b} \text{disproof}_v(u_i).
    \]
Revised PN-search($v$): algorithm

- **loop**: Compute or update $\text{proof}_v$ and $\text{disproof}_v$ numbers for each node in a bottom up fashion.
  - If $\text{proof}_v(\text{root}) = 0$ or $\text{disproof}_v(\text{root}) = 0$, then we are done, otherwise
    - $\triangleleft \text{proof}_v(\text{root}) \leq \text{disproof}_v(\text{root})$: we try to prove it.
    - $\triangleleft \text{proof}_v(\text{root}) > \text{disproof}_v(\text{root})$: we try to disprove it.

- $u \leftarrow \text{root}$; \{\text{* find the leaf to prove or disprove *}\}
  - if we try to prove, then
    - $\triangleleft$ while $u$ is not a leaf do
      - $\triangleleft$ if $u$ is a MAX node, then
        - $u \leftarrow$ leftmost child of $u$ with the smallest non-zero $\text{proof}_v$ number;
      - $\triangleleft$ if current is a MIN node, then
        - $u \leftarrow$ leftmost child of $u$ with a non-zero $\text{proof}_v$ number;
  - if we try to disprove, then
    - $\triangleleft$ while $u$ is not a leaf do
      - $\triangleleft$ if $u$ is a MAX node, then
        - $u \leftarrow$ leftmost child of $u$ with a non-zero $\text{disproof}_v$ number;
      - $\triangleleft$ if current is a MIN node, then
        - $u \leftarrow$ leftmost child of $u$ with the smallest non-zero $\text{disproof}_v$ number;

- Prove or disprove $u$; go to loop;
Multi-valued PN-search: algorithm

- When the values of the leaves are not binary, use an open value binary search to find an upper bound of the value.
  - Set the initial value of $v$ to be 1.
  - loop: PN-search($v$)
    - $\triangleright$ Prove the value of the search tree is $\geq v$ or disprove it by showing it is $< v$.
  - If it is proved, then double the value of $v$ and go to loop again.
  - If it is disproved, then the true value of the tree is between $\lfloor v/2 \rfloor$ and $v - 1$.
  - $\{ *$ Use a binary search to find the exact returned value of the tree. $* \}$
    - $low \leftarrow \lfloor v/2 \rfloor; high \leftarrow v - 1;$
    - while $low \leq high$ do
      - $\triangleright$ if $low = high$, then return $low$ as the tree value
      - $mid \leftarrow \lfloor (low + high)/2 \rfloor$
      - $PN$-search($mid$)
      - $\triangleright$ if it is disproved, then $high \leftarrow mid - 1$
      - $\triangleright$ else if it is proved, then $low \leftarrow mid$
Comments

- Can be used to construct opening books.
- Appears to be good for searching certain types of game trees.
  - Find the easiest way to prove or disprove a conjecture.
  - A dynamic strategy depends on work has been done so far.
- Performance has nothing to do with move ordering.
  - Performance of most previous algorithms depends heavily on whether a good move ordering can be found.
- Searching the “easiest” branch may not give you the best performance.
  - Performance depends on the value of each internal nodes.
- Commonly used in verifying conjectures, e.g., first-player win.
  - Partition the opening moves in a tree-like fashion.
  - Try to the “easiest” way to prove or disprove the given conjecture.
- Take into consideration the fact that some nodes may need more time to process than the other nodes.
References and further readings (1/2)

References and further readings (2/2)

- Bruce W. Ballard The *-minimax search procedure for trees containing chance nodes Artificial Intelligence, Volume 21, Issue 3, September 1983, Pages 327-350
- Kearns, Michael; Mansour, Yishay; Ng, Andrew Y. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. Machine Learning, 2002, 49.2-3: 193-208.