Chance Node Searching

Tsan-sheng Hsu

徐讚昇

tshsu@iis.sinica.edu.tw

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

Abstract

- Searching stochastic games
- Alpha-beta based techniques
  - Star0: exhaustive enumeration without cuts
  - Star0.5: cuts in between choices
  - Star1: cuts inside choices using bounds from an arbitrary move ordering
  - Star2: cuts inside choices using bounds from a good probing strategy
  - Star2.5: using an even better probing strategy
- MCTS based approaches
  - Sparse sampling

Stochastic games

- **Stochastic games have nodes whose outcome or move selections cannot be decided completely by players.**
  - **Pure stochastic: no action can be taken by a player before or after a random toss.**
    - ▷ *A dice game.*
  - **A priori chance node: a random toss is made first and then you make a decision based on the toss.**
    - ▷ *EinStein Würfelt Nicht (EWN) [Lorentz et al '12]: you make a random toss to decide what pieces that you can move, and then you make a move.*
  - **A posteriori chance node: you make a decision first and then followed by a random toss.**
    - ▷ *Chinese dark chess [Yen et al '14]: you pick a dark piece to flip, and then the piece is revealed decided by a random toss*

Searching stochastic games

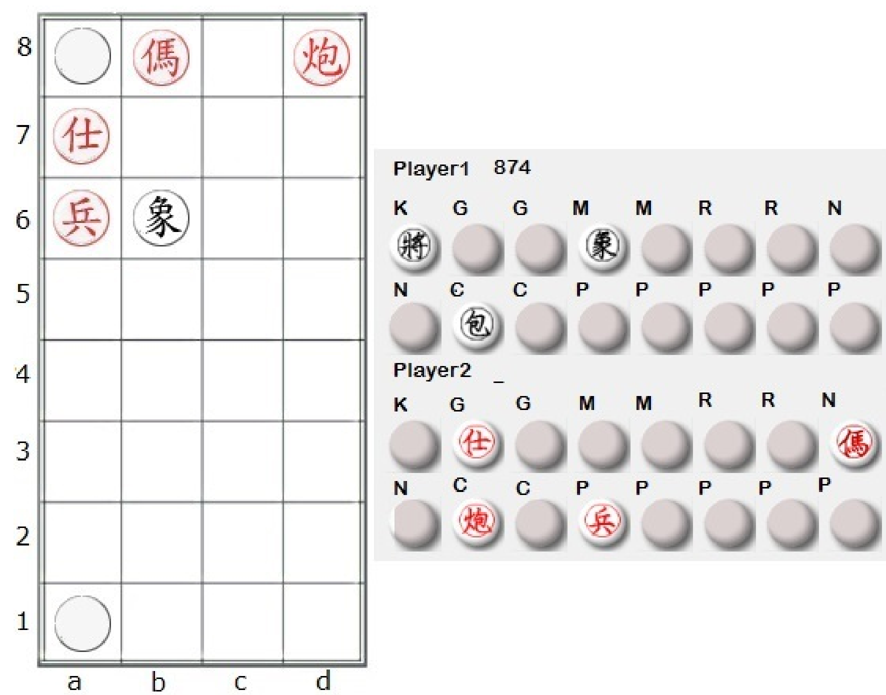
- **Because of a coin toss, the search space is greatly enlarged.**
  - **Example: In the opening phase, Chinese dark chess game tree has a very large branching factor.**
    - *After using reduction in symmetry, the first ply has $7*8$ possible outcomes.*
    - *The second ply has upto $14*31$ possible outcomes which is larger than 19x19 Go.*
- **Maybe need to compute all possible results from the coin toss to decide a good playing strategy.**
  - **The expected value of all possible outcomes is needed which may be difficult to apply any cuts.**

Search with chance nodes

- **Example: Chinese dark chess (CDC)**
  - **Two-player, zero sum**
  - **Complete information**
  - **Perfect information**
  - **Stochastic**
  - **There is a chance node during searching.**
    - *The value of a chance node is a distribution, not a fixed value.*
- **Previous work**
  - **Alpha-beta based [Ballard 1983]**
  - **Monte-Carlo based [Lancoto et al 2013] [Jouandeau and Cazenave '14]**

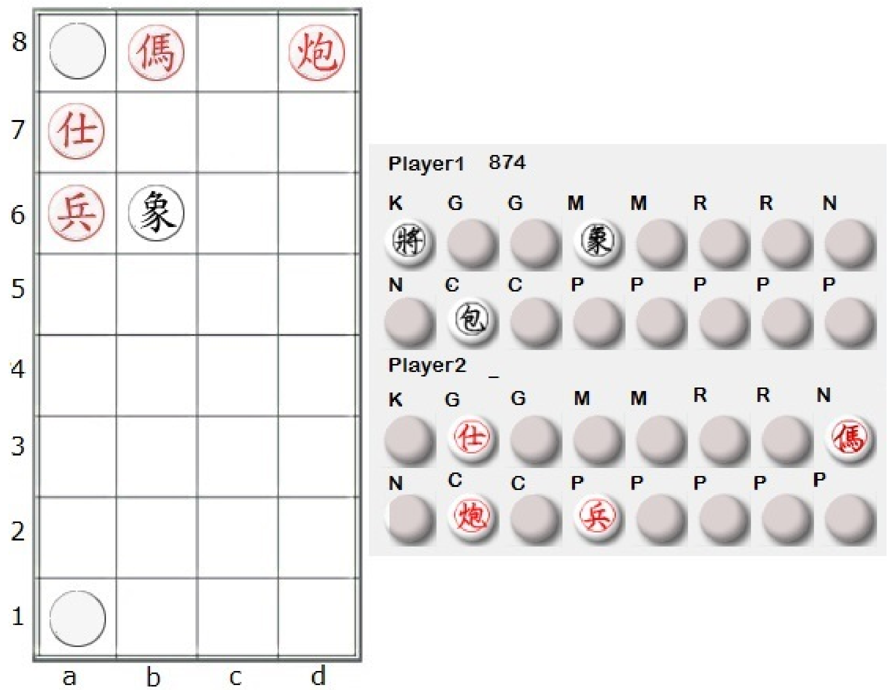
Example (1/4)

- It's BLACK turn and BLACK has 6 different possible legal moves which includes the four different moving made by its elephant and the two flipping moves at a1 or a8.
  - It is difficult for BLACK to secure a win by moving its elephant along any of the 3 possible directions, namely up, right or left, or by capturing the RED pawn at the left hand side.



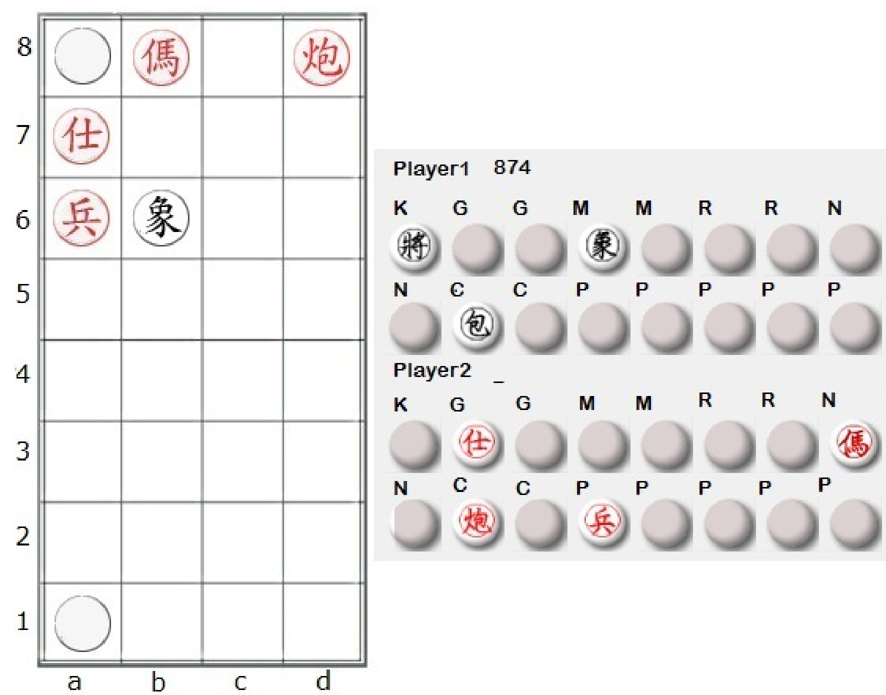
Example (2/4)

- **If BLACK flips a1, then there are 2 possible cases.**
  - **If a1 is BLACK cannon, then it is difficult for RED to win.**
    - ▷ *RED guard is in danger.*
  - **If a1 is BLACK king, then it is difficult for BLACK to lose.**
    - ▷ *BLACK king can go up through the right.*



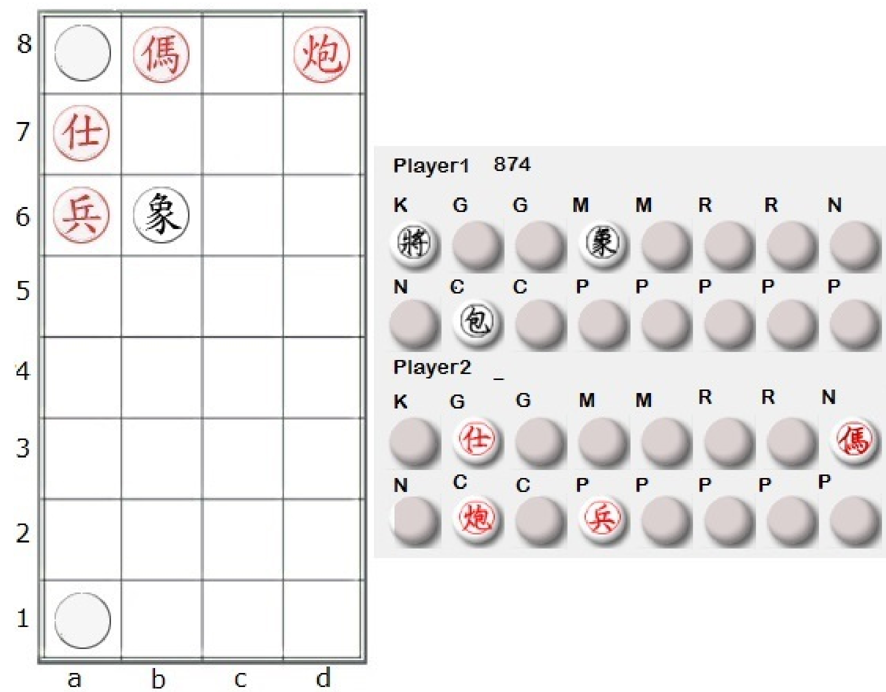
Example (3/4)

- **If BLACK flips a8, then there are 2 possible cases.**
  - **If a8 is BLACK cannon, then it is easy for RED to win.**
    - ▷ *RED cannon captures it immediately.*
  - **If a8 is BLACK king, then it is also easy for RED to win.**
    - ▷ *RED cannon captures it immediately.*



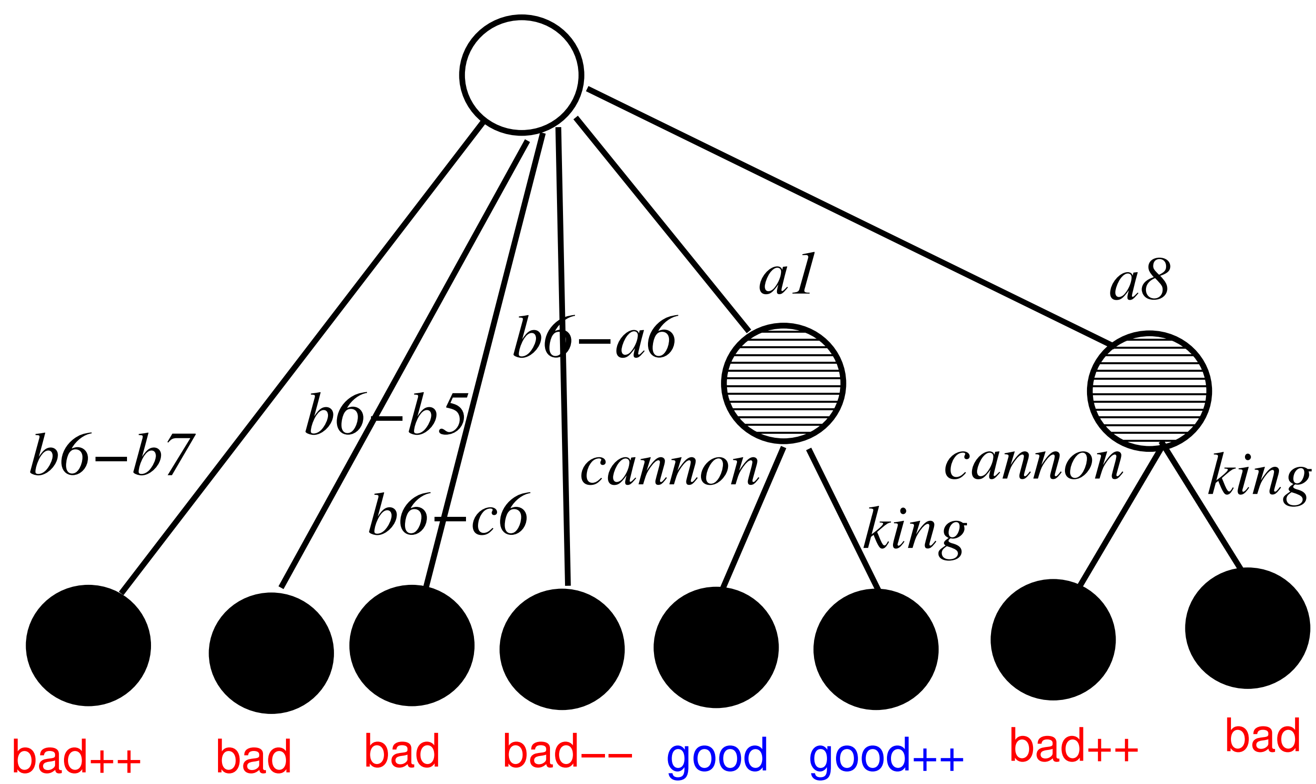
Example (4/4)

- **Conclusion:**
  - **It is vary bad for BLACK to flip a8.**
  - **It is bad for BLACK to move its elephant.**
  - **It is better for BLACK to flip a1.**



Example: illustration

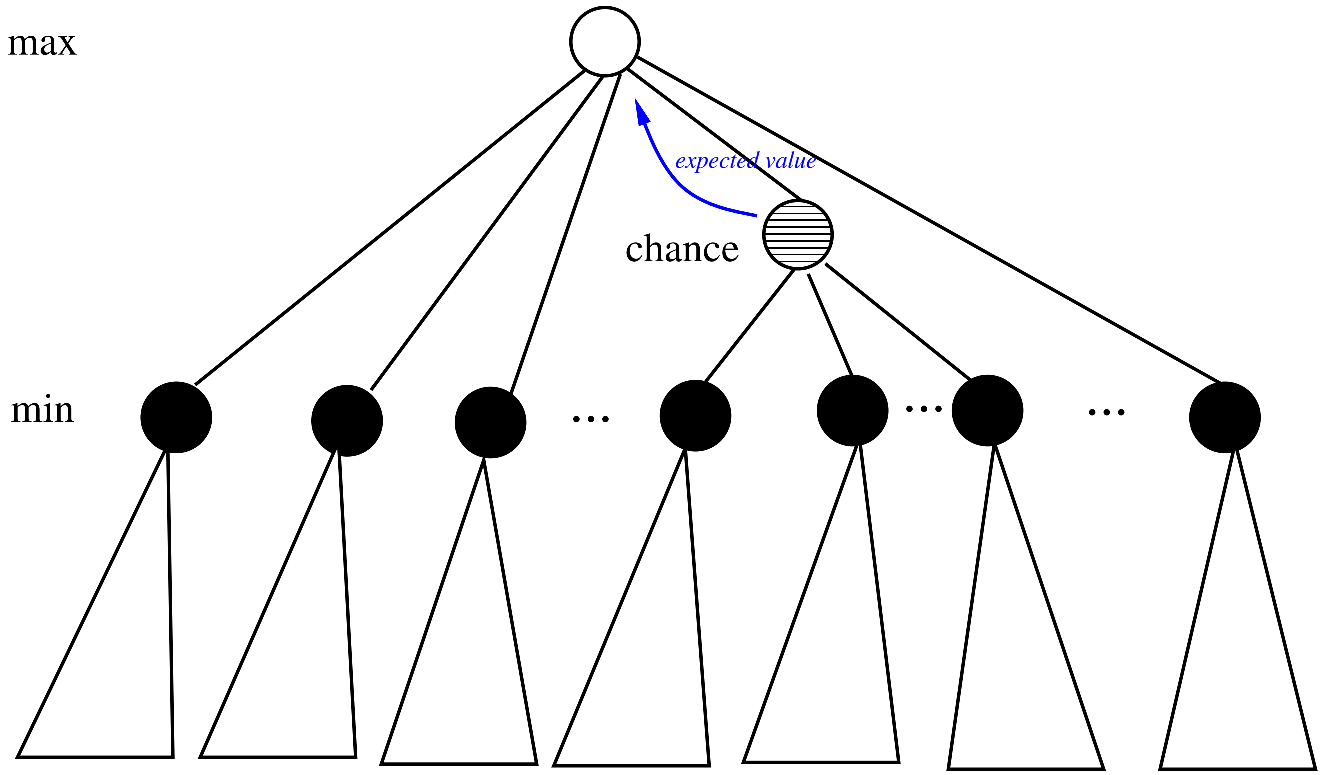
- **Conclusion:**
  - **It is vary bad for BLACK to flip a8.**
  - **It is bad for BLACK to move its elephant.**
  - **It is better for BLACK to flip a1.**



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 $E = \sum_{i=1}^{N} score(i) * Pr(x = i)$.**
  - **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, $N = 14 * 32$.**
    - ▷ ***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.***
    - ▷ $Pr(x = 1) = Pr(x = 8) = 1/14$.

Illustration



The probability distribution

- General case
  - Assume a chance node x has c choices $k_1, \ldots, k_c$.
  - The ith choice happens with the probability Pr_i.
    - $\sum_{i=1}^{c} Pr_i = 1$
- Special cases
  - Special case 1, called uniform (EQU): $Pr_i = 1/c$.
    - *All choices happen with an equal chance.*
    - *Example: EinStein Würfelt Nicht (EWN).*
  - Special case 2, called GCD: $Pr_i = w_i/D$ where each w_i is an integer and D is also an integer.
    - $D = \sum_{i=1}^{c} w_i$ *as in Chinese dark chess.*
- The above two special cases usually happen in game playing and can use the characteristics to do some optimization in arithmetic calculations.

Comments about EWN (1/3)

- $\sum_{i=1}^{c} Pr_i$ **is always** 1.
- **In EWN when there are only two pieces left, it appears that the above claim is not true.**
  - **Example 1: 1 and 6 with both probabilities being selected may look like** $\frac{5}{6}$.
    - *Assume the winning rates in example 1 are 0.75 and 0.23 for 1 and 6 being picked respectively.*
  - **Example 2: 1 and 2 may look like the probability of 1 being selected is** $\frac{1}{6}$, **but is** $\frac{5}{6}$ **for 2 being picked.**
    - *Assume the winning rates in example 2 are also 0.75 and 0.23 for 1 and 2 being picked respectively.*
- **Example 1 is favored over example 2 not because the sum of probabilities is larger!!!**

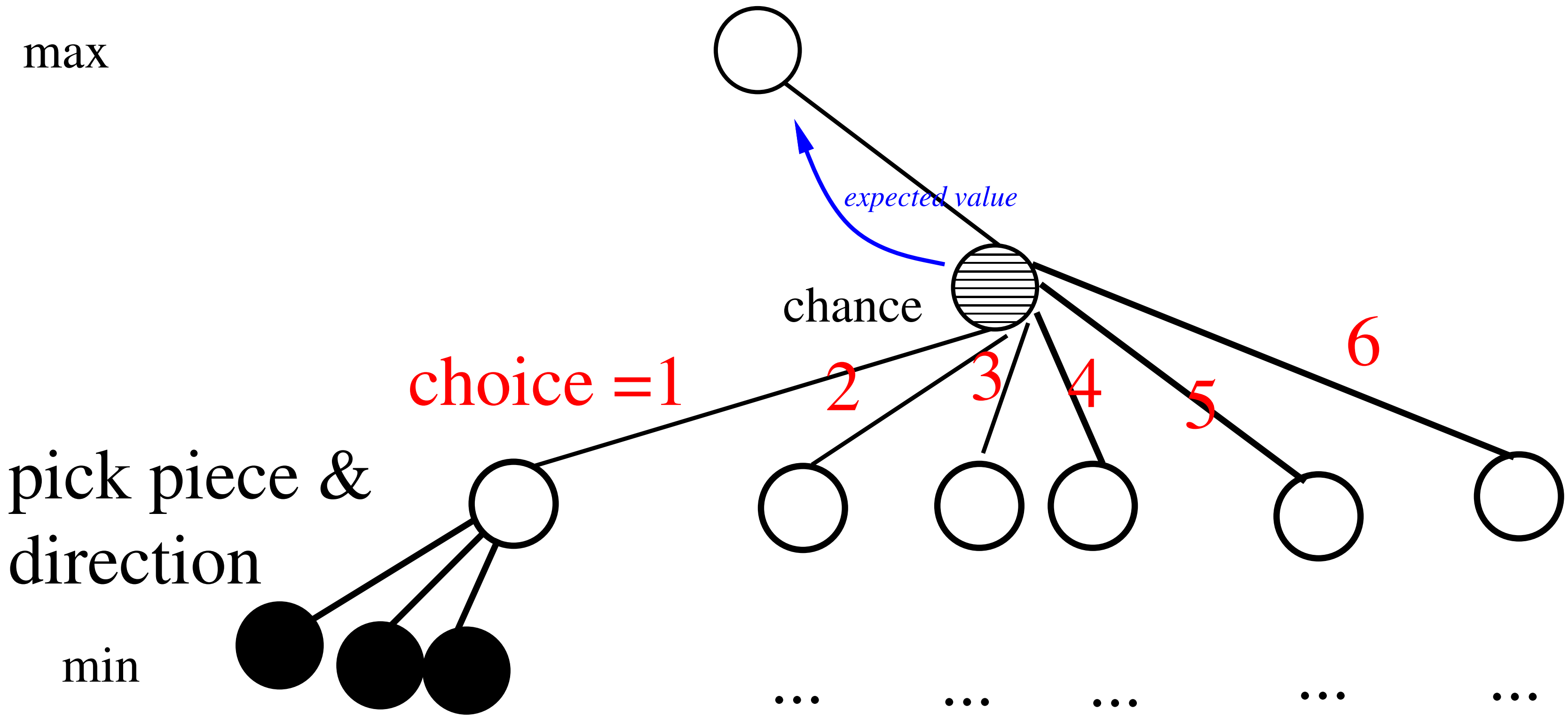
Comments about EWN (2/3)

- **EWN always has SIX choices.**
- **Example 1:**
  - **For choices 1 to 5, we can choose to move piece 1.**
  - **For choices 2 to 6, we can choose to move piece 6.**
  - **It appears that for choices 2 to 5, we have an equal chance of choosing either piece 1 or 6.**
    - ▷ ***However, due to the difference in winning rates, we always choose piece 1.***
  - **This means 1 is chosen with a probability of $\frac{5}{6}$ and 6 is picked with a probability of $\frac{1}{6}$.**
    - ▷ ***Hence the expected winning rate is*** $5 * \frac{1}{6} * 0.75 + 1 * \frac{1}{6} * 0.23$
- **Example 2:**
  - **For choice 1, we can choose to move piece 1.**
  - **For choices 2 to 6, we can choose to move piece 2.**
  - **This means 1 is chosen with a probability of $\frac{1}{6}$ and 2 is picked with a probability of $\frac{5}{6}$.**
    - ▷ ***Hence the expected winning rate is*** $1 * \frac{1}{6} * 0.75 + 5 * \frac{1}{6} * 0.23$

Comments about EWN (3/3)

- **Using transposition tables will help a lot in searching when some pieces are captured!!!**
- **Only ONE piece can be picked when choice = 1 or 6.**
- **If piece i is not being captured, then choice i can only pick that piece.**
- **For choices between 2 and 5, if the corresponding piece is being captured, then it has at most TWO pieces to choose from.**

Illustration for EWN



Algorithm: Chance_Search with Star0 (MAX)

- **Algorithm** $F3.0'$**(position** p**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// max node**
  - **determine the successor positions** $p_1, \ldots, p_b$
  - **if** $b = 0$ **// a terminal node**
    or $depth = 0$ **// remaining depth to search**
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - **then return** $f(p)$ **else begin**
    - ▷ $m := -\infty$
    - ▷ ***for*** $i := 1$ ***to*** b ***do***
    - ▷ ***begin***
    - ▷ ***if*** p_i ***is to play a chance node*** x
      then $t := Star0\_F3.0'(p_i, x, max\{alpha, m\}, beta, depth - 1)$
    - ▷ ***else*** $t := G3.0'(p_i, max\{alpha, m\}, beta, depth - 1)$
    - ▷ ***if*** $t > m$ ***then*** $m := t$
    - ▷ ***if*** $m \geq beta$ ***then return***(m) ***// beta cut off***
    - ▷ ***end***
  - **end;**
  - **return** m

Algorithm: Chance_Search with Star0 (MIN)

- **Algorithm** $G3.0'$(**position** p, **value** $alpha$, **value** $beta$, **integer** $depth$)
  - // min node
  - determine the successor positions $p_1, \ldots, p_b$
  - if $b = 0$ // a terminal node
    or $depth = 0$ // remaining depth to search
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - then return $f(p)$ else begin
    - ▷ $m := \infty$
    - ▷ *for* $i := 1$ *to* b *do*
    - ▷ *begin*
    - ▷ *if* p_i *is to play a chance node* x
      then $t := Star0\_G3.0'(p_i, x, alpha, min\{beta, m\}, depth - 1)$
    - ▷ *else* $t := F3.0'(p_i, alpha, min\{beta, m\}, depth - 1)$
    - ▷ *if* $t < m$ *then* $m := t$
    - ▷ *if* $m \leq alpha$ *then return*(m) // alpha cut off
    - ▷ *end*
  - end;
  - return m

Algorithm: $Star0$, uniform case (MAX)

- **version when all choices have equal probabilities**
- **max node**
- **Algorithm $Star0\_EQU\_F3.0'$(position p, node x, value $alpha$, value $beta$, integer $depth$)**
  - **// a max chance node x with c equal probability choices k_1, ..., k_c**
  - **// exhaustive search all possibilities and return the expected value**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - $vsum = 0$**; // initial sum of expected value**
  - **for $i = 1$ to c do**
  - **begin**
    - ▷ ***let p_i be the position of assigning k_i to x in p;***
    - ▷ $vsum$ += $G3.0'(p_i, -\infty, +\infty, depth)$;
  - **end**
- **return $vsum/c$; // return the expected score**

Algorithm: $Star0$, uniform case (MIN)

- version when all choices have equal probabilities
- min node
- Algorithm $Star0\_EQU\_G3.0'$(position p, node x, value $alpha$, value $beta$, integer $depth$)
  - // a min chance node x with c equal probability choices k_1, ..., k_c
  - // exhaustive search all possibilities and return the expected value
  - determine the possible values of the chance node x to be $k_1, \ldots, k_c$
  - $vsum = 0$; // initial sum of expected value
  - for $i = 1$ to c do
  - begin
    - ▷ *let p_i be the position of assigning k_i to x in p;*
    - ▷ $vsum$ += $F3.0'(p_i, -\infty, +\infty, depth)$;
  - end
- return $vsum/c$; // return the expected score

Star0: note

- $depth$ **stays the same in Star0 search since we are unwrapping a chance node.**
- **The search window from normal alpha-beta pruning cannot be applied in a chance node search since we are looking at the average of the outcome.**
    - **It is okay for one choice to have a very large or small value because it may be evened out by values from other choices.**
    - **Thus the search window is reset to** $(-\infty, \infty)$**.**

Algorithm: $Star0$, general case (MAX)

- **Algorithm** $Star0\_F3.0'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**,integer** $depth$**)**
  - **// a max chance node x with c choices k_1, ..., k_c**
  - **// the ith choice happens with the probability Pr$_i$**
  - **// exhaustive search all possibilities and return the expected value**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - $vexp = 0$; **// initial sum of expected value**
  - **for** $i = 1$ **to** c **do**
  - **begin**
    - ▷ ***let*** p_i ***be the position of assigning*** k_i ***to*** x ***in*** p***;***
    - ▷ $vexp$ ***+=*** ***Pr$_i$*** $*\ G3.0'(p_i, -\infty, +\infty, depth)$***;***
  - **end**
- **return** $vexp$**; // return the expected score**

Algorithm: $Star0$, general case (MIN)

- **Algorithm** $Star0\_G3.0'$(**position** p, **node** x, **value** $alpha$, **value** $beta$,**integer** $depth$)
  - // a min chance node x with c choices k_1, . . ., k_c
  - // the ith choice happens with the probability $\Pr_i$
  - // exhaustive search all possibilities and return the expected value
  - determine the possible values of the chance node x to be $k_1, \ldots, k_c$
  - $vexp = 0$; // initial sum of expected value
  - for $i = 1$ to c do
  - begin
    - ▷ *let p_i be the position of assigning k_i to x in p;*
    - ▷ $vexp$ += $\Pr_i * F3.0'(p_i, -\infty, +\infty, depth)$;
  - end
- **return** $vexp$; // return the expected score

Algorithm: $Star0$, GCD case (MAX)

- **Algorithm** $Star0\_GCD\_F3.0'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - // a max chance node x with c choices k_1, ..., k_c
  - // whose occurrence probability are w_1/D, ..., w_c/D
  - // and each w_i is an integer
  - // exhaustive search all possibilities and return the expected value
  - determine the possible values of the chance node x to be $k_1, \ldots, k_c$
  - $vsum = 0$; // initial sum of weight values
  - for $i = 1$ to c do
  - begin
    - ▷ *let* p_i *be the position of assigning* k_i *to* x *in* p*;*
    - ▷ $vsum$ += w_i * $G3.0'(p_i, -\infty, +\infty, depth)$;
  - end
- **return** $vsum/D$; // return the expected score

Algorithm: $Star0$, GCD case (MIN)

- **Algorithm** $Star0\_GCD\_G3.0'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - // a min chance node x with c choices k_1, ..., k_c
  - // whose occurrence probability are w_1/D, ..., w_c/D
  - // and each w_i is an integer
  - // exhaustive search all possibilities and return the expected value
  - determine the possible values of the chance node x to be $k_1, \ldots, k_c$
  - $vsum = 0$; // initial sum of weight values
  - for $i = 1$ to c do
  - begin
    - ▷ *let p_i be the position of assigning k_i to x in p;*
    - ▷ $vsum$ += w_i * $F3.0'(p_i, -\infty, +\infty, depth)$;
  - end
- **return** $vsum/D$; // return the expected score

Ideas for improvements

- **During a chance search, an exhaustive search method is used without any pruning.**
- **Ideas for further improvements**
  - **When some of the choices turn out very bad or good results, we know information about lower/upper bounds of the final value.**
  - **When you are in advantage, search for a bad choice first.**
    - ▷ *If the worst choice cannot is not too bad, then you can take this chance.*
  - **When you are in disadvantage, search for a good choice first.**
    - ▷ *If the best choice cannot is not good enough, then there is no need to take this chance.*
- **Examples: the average of 2 drawings of a dice is similar to a position with 2 choices with scores in [1..6].**
  - **The first drawing is 5. Then bounds of the average:**
    - ▷ *lower bound is 3*
    - ▷ *upper bound is 5.5.*
  - **The first drawing is 1. Then bounds of the average:**
    - ▷ *lower bound is 1*
    - ▷ *upper bound is 3.5.*

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, the ith choice is evaluated.**
  - **Assume $v_{min} \leq score(i) \leq v_{max}$.**
- **What are the lower and upper bounds, namely m_i and M_i, of the expected value of the chance node immediately after 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**
    - ▷ $m_1 \geq score(1) * Pr(x = 1) + v_{min} * (1 - Pr(x = 1))$, *and*
    - ▷ $M_1 \leq score(1) * Pr(x = 1) + v_{max} * (1 - Pr(x = 1))$.
  - $\cdots$
  - $i = i^*$, **we have computed $score(1), \ldots, score(i^*)$, and then know**
    - ▷ $m_{i^*} \geq \sum_{i=1}^{i^*} score(i) * Pr(x = i) + v_{min} * (1 - \sum_{i=1}^{i^*} Pr(x = i))$, *and*
    - ▷ $M_{i^*} \leq \sum_{i=1}^{i^*} score(i) * Pr(x = i) + v_{max} * (1 - \sum_{i=1}^{i^*} Pr(x = i))$.

Star0.5: uniform case (1/3)

- **For simplicity, let's assume $Pr(x = i) = \frac{1}{c}$, that is, the uniform case.**
- **For all i, and the evaluated value of the ith choice is v_i.**
- **Assume the search window entering a chance node with $N = c$ choices is $(alpha, beta)$.**
- **The value of a chance node after the first i choices are explored can be expressed as**
  - **an expected value $E_i = vsum_i/c$ obtained so far;**
    - ▷ $vsum_i = \sum_{j=1}^{i} v_j$
    - ▷ *This value is returned only when all choices are explored.*
      $\Rightarrow$ *The expected value of an un-explored child shouldn't be $\frac{v_{min}+v_{max}}{2}$.*
  - **a range of possible values $[m_i, M_i]$.**
    - ▷ $m_i = (\sum_{j=1}^{i} v_j + v_{min} \cdot (c - i))/c$
    - ▷ $M_i = (\sum_{j=1}^{i} v_j + v_{max} \cdot (c - i))/c$
  - **Invariants:**
    - ▷ $E_i \in [m_i, M_i]$
    - ▷ $E_c = m_c = M_c$

Star0.5: uniform case (2/3)

- **Let m_i and M_i be the current lower and upper bounds, respectively, of the expected value of this chance node immediately after the evaluation of the ith node.**
  - $m_i = (\sum_{j=1}^{i-1} v_j + v_i + v_{min} \cdot (c-i))/c$
  - $M_i = (\sum_{j=1}^{i-1} v_j + v_i + v_{max} \cdot (c-i))/c$
- **How to incrementally update m_i and M_i:**
  - $m_0 = v_{min}$
  - $M_0 = v_{max}$
  - $$m_i = m_{i-1} + (v_i - v_{min})/c \tag{1}$$
  - $$M_i = M_{i-1} + (v_i - v_{max})/c \tag{2}$$

Star0.5: uniform case (3/3)

- **Let m_i and M_i be the current lower and upper bounds, respectively, of the expected value of this chance node immediately after the evaluation of the ith node.**
  - $m_i = (\sum_{j=1}^{i-1} v_j + v_i + v_{min} \cdot (c - i))/c$
  - $M_i = (\sum_{j=1}^{i-1} v_j + v_i + v_{max} \cdot (c - i))/c$
- **The current search window is $(alpha, beta)$.**
  - **No more searching is needed when**
    - ▷ $m_i \geq beta$, ***chance node cut off I;***
      $\Rightarrow$ ***The lower bound found so far is good enough.***
      $\Rightarrow$ ***Similar to a beta cut off.***
      $\Rightarrow$ ***The returned value is m_i.***
    - ▷ $M_i \leq alpha$, ***chance node cut off II.***
      $\Rightarrow$ ***The upper bound found so far is bad enough.***
      $\Rightarrow$ ***Similar to an alpha cut off.***
      $\Rightarrow$ ***The returned value is M_i.***

Example for Star0.5

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

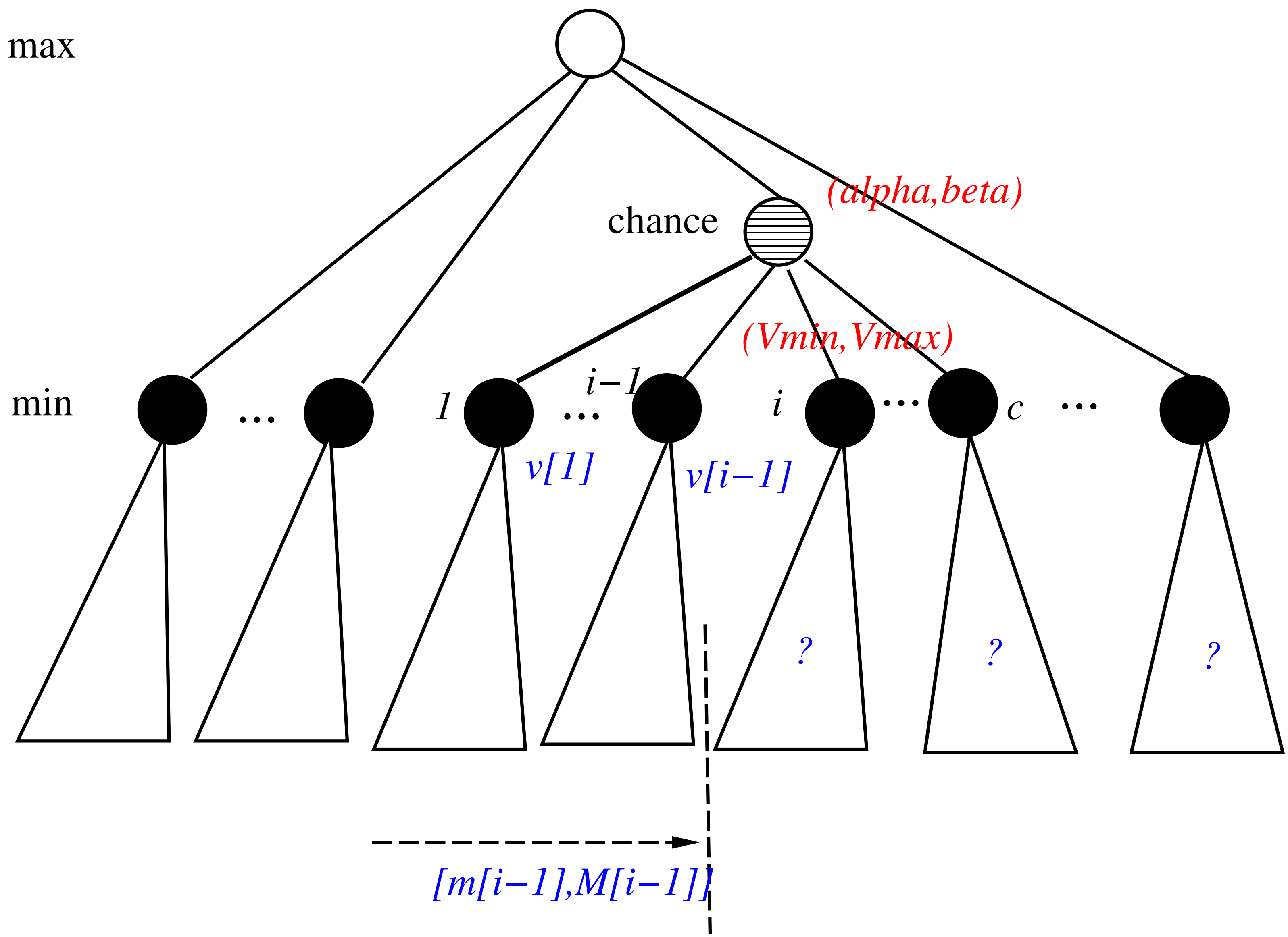
Star0.5: uniform case (MAX)

- **Algorithm** $Star0.5\_EQU\_F3.0'$(**position** p, **node** x, **value** $alpha$, **value** $beta$, **integer** $depth$)
  - // a max chance node x with c equal probability choices k_1, ..., k_c
  - determine the possible values of the chance node x to be $k_1, \ldots, k_c$
  - $m_0 = v_{min}$, $M_0 = v_{max}$ // initial lower and upper bounds
  - $vsum = 0$; // initial sum of expected values
  - for $i = 1$ to c do
  - begin
    - ▷ *let p_i be the position of assigning k_i to x in p;*
    - ▷ *t := $G3.0'(p_i, v_{min}, v_{max}, depth)$*
    - ▷ *$m_i = m_{i-1} + (t - v_{min})/c$, $M_i = M_{i-1} + (t - v_{max})/c$; // update the bounds*
    - ▷ *if $m_i \geq beta$ then return m_i; // failed high, chance node cut off I*
    - ▷ *if $M_i \leq alpha$ then return M_i; // failed low, chance node cut off II*
    - ▷ *$vsum$ += t;*
  - end
- **return** $vsum/c$;

Star0.5: uniform case (MIN)

- **Algorithm** $Star0.5\_EQU\_G3.0'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// a min chance node x with c equal probability choices k_1, ..., k_c**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - $m_0 = v_{min}$, $M_0 = v_{max}$ **// initial lower and upper bounds**
  - $vsum = 0$; **// initial sum of expected values**
  - **for** $i = 1$ **to** c **do**
  - **begin**
    - ▷ ***let*** p_i ***be the position of assigning*** k_i ***to*** x ***in*** p***;***
    - ▷ $t := F3.0'(p_i, v_{min}, v_{max}, depth)$
    - ▷ $m_i = m_{i-1} + (t - v_{min})/c$, $M_i = M_{i-1} + (t - v_{max})/c$; ***// update the bound***
    - ▷ ***if*** $m_i \geq beta$ ***then return*** m_i***; // failed high, chance node cut off I***
    - ▷ ***if*** $M_i \leq alpha$ ***then return*** M_i***; // failed low, chance node cut off II***
    - ▷ $vsum \mathrel{+}= t$;
  - **end**
- **return** $vsum/c$**;**

Illustration: Star0.5



Ideas for further improvements (1/2)

- **The above two cut offs comes from each time a choice is completely searched.**
  - **When $m_i \geq beta$, chance node cut off I,**
    - ▷ ***which means*** $(\sum_{j=1}^{i-1} v_j + v_i + v_{min} \cdot (c-i))/c \geq beta$.
  - **When $M_i \leq alpha$, chance node cut off II,**
    - ▷ ***which means*** $(\sum_{j=1}^{i-1} v_j + v_i + v_{max} \cdot (c-i))/c \leq alpha$.
- **Further cut off can be obtained before during searching a choice.**
  - **Assume after searching the first $i-1$ choices, no chance node cut off happens.**
  - **Before searching the ith choice, we know that if v_i is large enough, then it will raise the lower bound of the chance node which may trigger a chance node cut off I.**
  - **How large should v_i be for this to happen?**
    - ▷ ***chance node cut off I:*** $(\sum_{j=1}^{i-1} v_j + v_i + v_{min} \cdot (c-i))/c \geq beta$
    - ▷ $\Rightarrow v_i \geq B_{i-1} = c \cdot beta - (\sum_{j=1}^{i-1} v_j - v_{min} * (c-i))$
    - ▷ B_{i-1} ***is the threshold for cut off I to happen.***

Ideas for further improvements (2/2)

- **Similarly,**
  - **Assume after searching the first $i-1$ choices, no chance node cut off happens.**
  - **Before searching the ith choice, we know that if v_i is small enough, then it will lower the upper bound of the chance node which may trigger a chance node cut off II.**
  - **How small should v_i be for this to happen?**
    - ▷ ***chance node cut off II:*** $(\sum_{j=1}^{i-1} v_j + v_i + v_{max} \cdot (c-i))/c \leq alpha$
    - ▷ $\Rightarrow v_i \leq A_{i-1} = c \cdot alpha - (\sum_{j=1}^{i-1} v_j - v_{max} * (c-i))$
    - ▷ A_{i-1} ***is the threshold for cut off II to happen.***

Example: Star1

- **Example: the average of 2 drawings of a dice is similar to a position with 2 choices with scores in [1..6].**
  - $[m_0, M_0] = [v_{min}, v_{max}] = [1, 6]$
  - **Assume** $(alpha, beta) = (3.25, 3.95)$
- **The first drawing $v_1 = 3$. Then bounds of the average:**
  - **lower bound is 2; upper bound is 4.5.**
  - $[m_1, M_1] = [2, 4.5]$
- **Before the second drawing, the search will**
  - **failed-low if** $\frac{v_2+3}{2} \leq alpha = 3.25$ **which means the search fails low if** $v_2 \leq 3.5$**.**
  - **failed-high if** $\frac{v_2+3}{2} \geq beta = 3.95$ **which means the search fails high if** $v_2 \geq 4.9$**.**
- **Hence we can set the search window for the second search to be $(3.5, 4.9)$ instead of $[1, 6]$.**
  - ▷ ***We only need to do a test on whether v_2 is 4 or not.***

Formulas for the uniform case: Star1

- **Set the window for searching the ith choice to be (A_{i-1}, B_{i-1}) which means no further search is needed if the result is not within this window.**
  - (A_{i-1}, B_{i-1}) **is the window for searching the ith choice instead of using $(alpha, beta)$.**
- **How to incrementally update A_i and B_i?**
  - $$A_0 = c \cdot (alpha - v_{max}) + v_{max} \tag{3}$$
  - $$B_0 = c \cdot (beta - v_{min}) + v_{min} \tag{4}$$
  - $$A_i = A_{i-1} + v_{max} - v_i \tag{5}$$
  - $$B_i = B_{i-1} + v_{min} - v_i \tag{6}$$
- **Comment:**
  - **May want to use zero-window search to test first.**

Algorithm: Chance_Search with Star1 (MAX)

- **Algorithm** $F3.1'$**(position** p**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// max node**
  - **determine the successor positions** $p_1, \ldots, p_b$**;**
  - **if** $b = 0$ **// a terminal node**
    or $depth = 0$ **// remaining depth to search**
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - **then return** $f(p)$**; else begin**
    - ▷ $m := -\infty$;
    - ▷ ***for*** $i := 1$ ***to*** b ***do***
    - ▷ ***begin***
    - ▷   ***if*** p_i ***is to play a chance node*** x
          then $t := Star1\_F3.1'(p_i, x, \max\{alpha, m\}, beta, depth - 1)$;
    - ▷   ***else*** $t := G3.1'(p_i, \max\{alpha, m\}, beta, depth - 1)$;
    - ▷   ***if*** $t > m$ ***then*** $m := t$;
    - ▷   ***if*** $m \geq beta$ ***then return***(m); ***// beta cut off***
    - ▷ ***end;***
  - **end;**
  - **return** m**;**

Algorithm: Chance_Search with Star1 (MIN)

- **Algorithm** $G3.1'$**(position** p**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// min node**
  - **determine the successor positions** $p_1, \ldots, p_b$**;**
  - **if** $b = 0$ **// a terminal node**
    or $depth = 0$ **// remaining depth to search**
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - **then return** $f(p)$**; else begin**
    - *m* $:= \infty$;
    - ***for*** $i := 1$ ***to*** b ***do***
    - ***begin***
    -     ***if*** p_i ***is to play a chance node*** x
          then $t := Star1\_G3.1'(p_i, x, alpha, \min\{beta, m\}, depth - 1)$;
    -     ***else*** $t := F3.1'(p_i, alpha, \min\{beta, m\}, depth - 1)$;
    -     ***if*** $t < m$ ***then*** $m := t$;
    -     ***if*** $m \leq alpha$ ***then return***(m); ***// alpha cut off***
    - ***end;***
  - **end;**
  - **return** m**;**

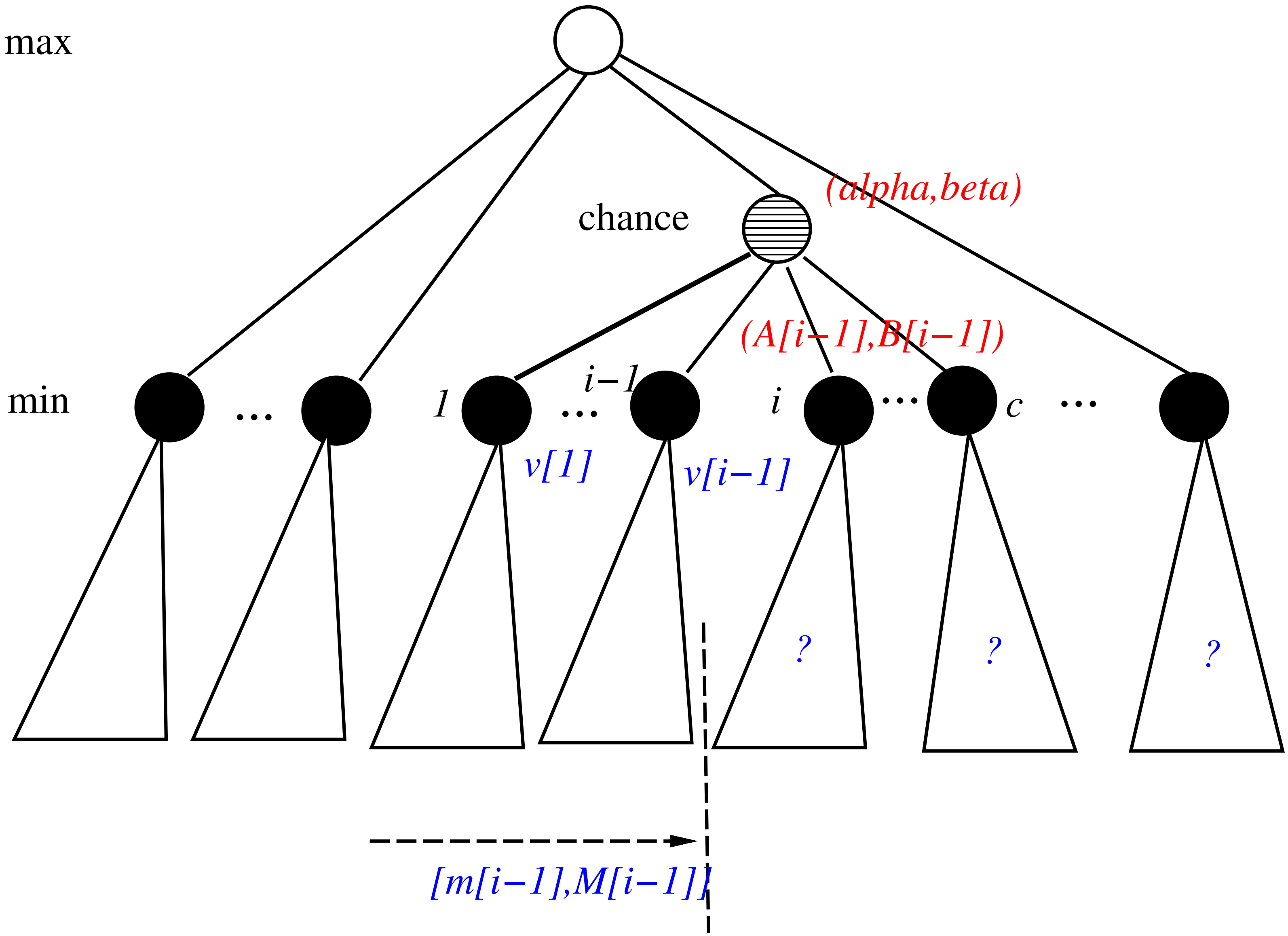
Star1: uniform case (MAX)

- **Algorithm** $Star1\_EQU\_F3.1'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// a max chance node x with c equal probability choices k_1, . . ., k_c**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - $A_0 = c \cdot (alpha - v_{max}) + v_{max}$, $B_0 = c \cdot (beta - v_{min}) + v_{min}$;
  - $m_0 = v_{min}$, $M_0 = v_{max}$ **// initial lower and upper bounds**
  - $vsum = 0$; **// initial sum of expected values**
  - **for** $i = 1$ **to** c **do**
  - **begin**
    - ▷ ***let** p_i **be the position of assigning** k_i **to** x **in** p;*
    - ▷ $t := G3.1'(p_i, \max\{A_{i-1}, v_{min}\}, \min\{B_{i-1}, v_{max}\}, depth)$
    - ▷ $m_i = m_{i-1} + (t - v_{min})/c$, $M_i = M_{i-1} + (t - v_{max})/c$;
    - ▷ ***if** $t \geq B_{i-1}$ **then return** m_i; **// failed high, chance node cut off I***
    - ▷ ***if** $t \leq A_{i-1}$ **then return** M_i; **// failed low, chance node cut off II***
    - ▷ $vsum \mathrel{+}= t$;
    - ▷ $A_i = A_{i-1} + v_{max} - t$, $B_i = B_{i-1} + v_{min} - t$;
  - **end**
- **return** $vsum/c$;

Star1: uniform case (MIN)

- **Algorithm** $Star1\_EQU\_G3.1'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// a min chance node x with c equal probability choices k_1, $\ldots$, k_c**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - $A_0 = c \cdot (alpha - v_{max}) + v_{max}$, $B_0 = c \cdot (beta - v_{min}) + v_{min}$;
  - $m_0 = v_{min}$, $M_0 = v_{max}$ **// initial lower and upper bounds**
  - $vsum = 0$; **// initial sum of expected values**
  - **for** $i = 1$ **to** c **do**
  - **begin**
    - ▷ ***let*** p_i ***be the position of assigning*** k_i ***to*** x ***in*** p;
    - ▷ $t := F3.1'(p_i, \textbf{max}\{A_{i-1}, v_{min}\}, \textbf{min}\{B_{i-1}, v_{max}\}, depth)$
    - ▷ $m_i = m_{i-1} + (t - v_{min})/c$, $M_i = M_{i-1} + (t - v_{max})/c$;
    - ▷ ***if*** $t \geq B_{i-1}$ ***then return*** m_i; ***// failed high, chance node cut off I***
    - ▷ ***if*** $t \leq A_{i-1}$ ***then return*** M_i; ***// failed low, chance node cut off II***
    - ▷ $vsum \mathrel{+}= t$;
    - ▷ $A_i = A_{i-1} + v_{max} - t$, $B_i = B_{i-1} + v_{min} - t$;
  - **end**
- **return** $vsum/c$;

Illustration: Star1



Star1: general case (1/3)

- **Assume the search window entering a chance node with $N = c$ choices is $(alpha, beta)$.**
- **The ith choice happens with the probability $Pr(x = i) = Pr_i$.**
- **For all i, the evaluated value of the ith choice is v_i.**
- **The value of a chance node after the first i choices are explored can be expressed as**
  - **an expected value $E_i = vexp_i$;**
    - ▷ $vexp_i = \sum_{j=1}^{i} Pr_j * v_j$
    - ▷ ***This value is returned only when all choices are explored.***
      $\Rightarrow$ ***The expected value of an un-explored child shouldn't be $\frac{v_{min}+v_{max}}{2}$.***
  - **a range of possible values $[m_i, M_i]$.**
    - ▷ $m_i = vexp_i + \sum_{j=i+1}^{c} Pr_j * v_{min}$
    - ▷ $M_i = vexp_i + \sum_{j=i+1}^{c} Pr_j * v_{max}$
  - **Invariants:**
    - ▷ $E_i \in [m_i, M_i]$
    - ▷ $E_c = m_c = M_c$

Star1: general case (2/3)

- **Let m_i and M_i be the current lower and upper bounds, respectively, of the expected value of this chance node immediately after the evaluation of the ith node.**
  - $m_i = vexp_{i-1} + Pr_i * v_i + \sum_{j=i+1}^{c} Pr_j * v_{min}$
  - $M_i = vexp_{i-1} + Pr_i * v_i + \sum_{j=i+1}^{c} Pr_j * v_{max}$
- **How to incrementally update m_i and M_i:**
  - $m_0 = v_{min}$
  - $M_0 = v_{max}$
  - $$m_i = m_{i-1} + Pr_i * (v_i - v_{min}) \quad (7)$$
  - $$M_i = M_{i-1} + Pr_i * (v_i - v_{max}) \quad (8)$$

Star1: general case (3/3)

- **The current search window is $(alpha, beta)$.**
- **No more searching is needed when**
  - $m_i \geq beta$, **chance node cut off I;**
    $\Rightarrow$ The lower bound found so far is good enough.
    $\Rightarrow$ Similar to a beta cut off.
    $\Rightarrow$ The returned value is m_i.
  - $M_i \leq alpha$, **chance node cut off II.**
    $\Rightarrow$ The upper bound found so far is bad enough.
    $\Rightarrow$ Similar to an alpha cut off.
    $\Rightarrow$ The returned value is M_i.

Star1 cut off: general case (1/2)

- **When** $m_i \geq beta$, **chance node cut off I,**
  - **which means** $vexp_{i-1} + Pr_i * v_i + \sum_{j=i+1}^{c} Pr_j * v_{min} \geq beta$
  - $\Rightarrow v_i \geq B_{i-1} = \frac{1}{Pr_i} \cdot (beta - (vexp_{i-1} + \sum_{j=i+1}^{c} Pr_j * v_{min}))$
- **When** $M_i \leq alpha$, **chance node cut off II,**
  - **which means** $vexp_{i-1} + Pr_i * v_i + \sum_{j=i+1}^{c} Pr_j * v_{max} \leq alpha$
  - $\Rightarrow v_i \leq A_{i-1} = \frac{1}{Pr_i} \cdot (alpha - (vexp_{i-1} + \sum_{j=i+1}^{c} Pr_j * v_{max}))$
- **Hence set the window for searching the** i**th choice to be** (A_{i-1}, B_{i-1}) **which means no further search is needed if the result is not within this window.**

Star1 cut off: general case (2/2)

- **How to incrementally update A_i and B_i?**
  - $$A_0 = \frac{1}{Pr_1} \cdot (alpha - v_{max} * \sum_{i=1}^{c} Pr_i) + v_{max} \tag{9}$$
  - $$B_0 = \frac{1}{Pr_1} \cdot (beta - v_{min} * \sum_{i=1}^{c} Pr_i) + v_{min} \tag{10}$$
  - $$A_i = \frac{1}{Pr_{i+1}} * (Pr_i * A_{i-1} + Pr_{i+1} * v_{max} - Pr_i * v_i) \tag{11}$$
  - $$B_i = \frac{1}{Pr_{i+1}} * (Pr_i * B_{i-1} + Pr_{i+1} * v_{min} - Pr_i * v_i) \tag{12}$$

Star1: general case (MAX)

- **Algorithm** $Star1\_F3.1'$**(position** p, **node** x, **value** $alpha$, **value** $beta$, **integer** $depth$**)**
  - **// a max chance node x with c choices k_1, ..., k_c**
  - **// the ith choice happens with the probability Pr$_i$**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - **initialize A_0 and B_0 using formulas (9) and (10)**
  - $m_0 = v_{min}$, $M_0 = v_{max}$ **// initial lower and upper bounds**
  - $vexp = 0$; **// initial weighted sum of expected values**
  - **for** $i = 1$ **to** c **do**
  - **begin**
    - ▷ ***let*** P_i ***be the position of assigning*** k_i ***to*** x ***in*** p***;***
    - ▷ $t := G3.1'(p_i,$**max**$\{A_{i-1},v_{min}\}$,**min**$\{B_{i-1},v_{max}\},depth)$
    - ▷ ***incrementally update*** m_i ***and*** M_i ***using formulas (7) and (8)***
    - ▷ ***if*** $t \geq B_{i-1}$ ***then return*** m_i; ***// failed high, chance node cut off I***
    - ▷ ***if*** $t \leq A_{i-1}$ ***then return*** M_i; ***// failed low, chance node cut off II***
    - ▷ $vexp$ += $Pr_i * t$;
    - ▷ ***incrementally update*** A_i ***and*** B_i ***using formulas (11) and (12)***
  - **end**
- **return** $vexp$;

Star1: general case (MIN)

- **Algorithm** $Star1\_G3.1'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// a min chance node x with c choices k_1, ..., k_c**
  - **// the ith choice happens with the probability Pr$_i$**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - **initialize A_0 and B_0 using formulas (9) and (10)**
  - $m_0 = v_{min}$, $M_0 = v_{max}$ **// initial lower and upper bounds**
  - $vexp = 0$; **// initial weighted sum of expected values**
  - **for** $i = 1$ **to** c **do**
  - **begin**
    - ▷ ***let P_i be the position of assigning k_i to x in p;***
    - ▷ $t := F3.1'(p_i$,**max**$\{A_{i-1}$,$v_{min}\}$,**min**$\{B_{i-1}$,$v_{max}\}$,$depth)$
    - ▷ ***incrementally update m_i and M_i using formulas (7) and (8)***
    - ▷ ***if*** $t \geq B_{i-1}$ ***then return*** m_i; ***// failed high, chance node cut off I***
    - ▷ ***if*** $t \leq A_{i-1}$ ***then return*** M_i; ***// failed low, chance node cut off II***
    - ▷ $vexp \mathrel{+}= Pr_i * t$;
    - ▷ ***incrementally update A_i and B_i using formulas (11) and (12)***
  - **end**
- **return** $vexp$**;**

Star1: GCD case (1/2)

- **Assume the ith choice happens with a chance w_i/c where $c = \sum_{i=1}^{N} w_i$ and N is the total number of choices.**
  - $m_0 = v_{min}$
  - $M_0 = v_{max}$
  - $m_i = (\sum_{j=1}^{i-1} w_j \cdot v_j + w_i \cdot v_i + v_{min} \cdot (c - \sum_{j=1}^{i} w_j))/c$
    - ▷
$$m_i = m_{i-1} + (w_i/c) \cdot (v_i - v_{min}) \tag{13}$$
  - $M_i = (\sum_{j=1}^{i-1} w_j \cdot v_j + w_i \cdot v_i + v_{max} \cdot (c - \sum_{j=1}^{i} w_j))/c$
    - ▷
$$M_i = M_{i-1} + (w_i/c) \cdot (v_i - v_{max}) \tag{14}$$

Star1: GCD case (2/2)

- **Assume the ith choice happens with a chance w_i/c where $c = \sum_{i=1}^{N} w_i$ and N is the total number of choices.**
  - $$A_0 = (c/w_1) \cdot (alpha - v_{max}) + v_{max} \quad (15)$$
  - $$B_0 = (c/w_1) \cdot (beta - v_{min}) + v_{min} \quad (16)$$
  - $A_{i-1} = (c \cdot alpha - (\sum_{j=1}^{i-1} w_j \cdot v_j - v_{max} \cdot (c - \sum_{j=1}^{i} w_j)))/w_i$
    - $$A_i = (w_i/w_{i+1}) \cdot (A_{i-1} - v_i) + v_{max} \quad (17)$$
  - $B_{i-1} = (c \cdot beta - (\sum_{j=1}^{i-1} w_j \cdot v_j - v_{min} \cdot (c - \sum_{j=1}^{i} w_j)))/w_i$
    - $$B_i = (w_i/w_{i+1}) \cdot (B_{i-1} - v_i) + v_{min} \quad (18)$$

Remarks

- **To know what operations are simplified from the general case to special cases, compare these formulas**

	general case	GCD case	uniform case
m_i	7	13	1
M_i	8	14	2
a_0	9	15	3
b_0	10	16	4
A_i	11	17	5
B_i	12	18	6

Comments (1/2)

- Star0.5 finishes searching a choice using the maximum window size and then decide whether to go on searching the next choice or not, where Star1 can use sharper window size to end searching a choice earlier.
- We illustrate the ideas using a fail soft version of the alpha-beta algorithm.
  - Original and fail hard version have a simpler logic in maintaining the search interval.
  - The semantic of comparing an exact return value with an expected returning value is something that needs careful thinking.
  - May want to pick a chance node with a lower expected value but having a hope of winning, not one with a slightly higher expected value but having no hope of winning when you are in disadvantageous.
  - May want to pick a chance node with a lower expected value but having no chance of losing, not one with a slightly higher expected value but having a chance of losing when you are in advantage.
  - Do not always pick one with a slightly larger expected value. Give the second one some chance to be selected.

Comments (2/2)

- **Need to revise algorithms carefully when dealing with the original, fail hard or NegaScout version.**
  - **What does it mean to combine bounds from a fail hard version?**
- **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.**
  - **Not only we can terminate the searching of choices earlier, but also we can terminate the searching of a particular choice earlier.**
- **Exist other improvements by searching choices of a chance node "in parallel".**

Implementation hints (1/2)

- **Fully unwrap a chance node takes more time than that of a non-chance node.**
  - **If you set your depth limit to d for a game without chance nodes, then the depth limit should be lower for that game when chance node is introduced.**
  - **Technically speaking, a chance node adds at least one level.**
    - ▷ ***Depending on the number of choices you have compared to the number of non-chance children, you may need to reduce the search depth limit by at least 3 or 5, and maybe 7.***
    - ▷ ***Estimate the complexity of a chance node by comparing the number of choices of a chance node and the number of non-chance-node moves.***
- **Without searching a chance node, it is easy to obtain not enough progress by just searching a long sequence of non-chance nodes.**
  - **In CDC, when there are only a limited number of revealed pieces, there is not much you can do by just moving around.**

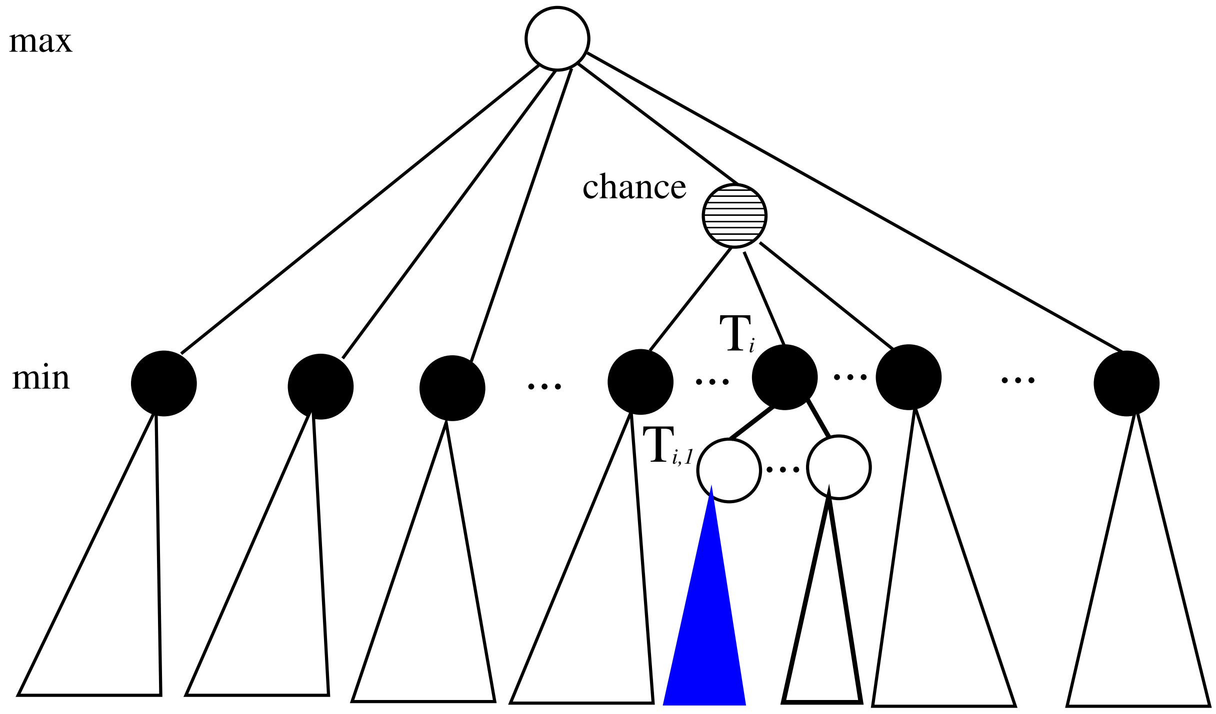
Implementation hints (2/2)

- Practical considerations, for example in Chinese Dark Chess (CDC), are as follows.
  - You normally do not need to consider the consequence of flipping more than 2 dark pieces.
    - ▷ *Set a maximum number of chance node searching in any DFS search path.*
  - It makes little sense to consider ending a search with exploring a chance node.
    - ▷ *When depth limit left is less than 3 or 4, stop exploring chance nodes.*
  - It also makes little sense to consider the consequence of exploring 2 chance nodes back to back.
    - ▷ *Make sure two chance nodes in a DFS search path is separated by at least 3 or 4 non-chance nodes.*
  - It is rarely the case that a chance node exploration is the first ply to consider in move ordering unless it is recommended by a prior knowledge or no other non-chance-node moves exists.

More ideas for improvements

- **Notations**
  - **Assume p is a chance node with the tree T.**
    - ▷ ***T_i is the tree of p when for the ith choice.***
    - ▷ ***$T_{i,j}$ is the jth branch of T_i, namely, with the root $p_{i,j}$.***
    - ▷ ***v_i is the evaluated value of T_i.***
    - ▷ ***$v_{i,j}$ is the evaluated value of $T_{i,j}$.***
- **An exact probe of a tree rooted at r is thus to fully search a subtree rooted at a child of r.**
  - ▷ ***An exact probe of T is thus to fully search T_i for some i and then obtain v_i.***
  - ▷ ***An exact probe of T_i is to fully search $T_{i,j}$ for some j and then obtain $v_{i,j}$.***
- **Can do better by not searching the DFS order.**
  - **It is not necessary to search completely T_1 and then start to look at the subtree of T_2, ... etc.**
    - ▷ ***The approach used by Star1.***
  - **Probe T_i gives you some information about the possible range of v_i.**

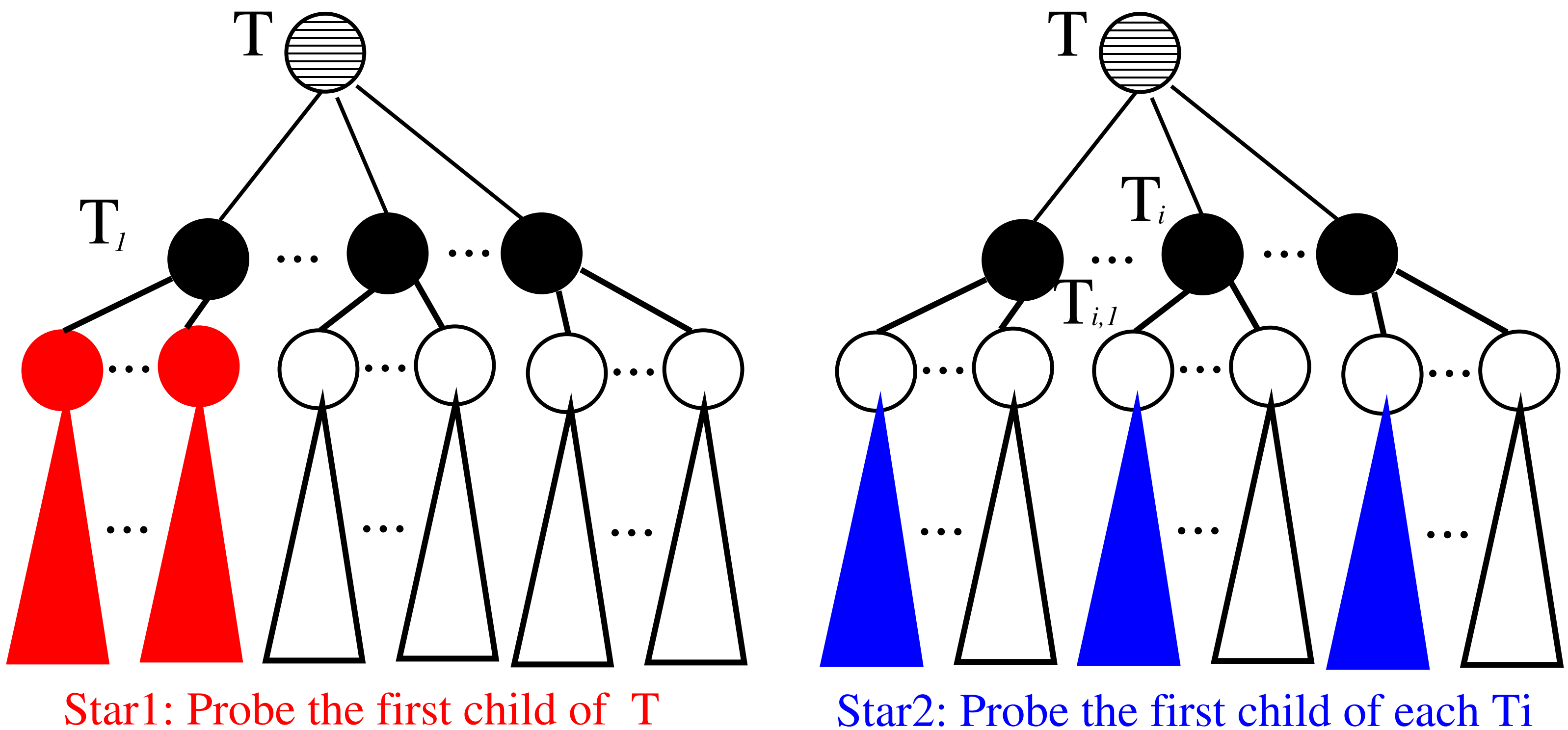
Illustration: Probe



Star2: MAX node, general case

- p is a MAX node. Thus each child p_i is a MIN node.
- We have probed the first child of T_i and obtained $v_{i,1}$.
  - Since p_i is a MIN node, $v_{i,1}$ is an upper bound of v_i which is usually lower than the maximum possible value v_{max}.
  - The upper bound of v_1 is thus lowered.
  - It is possible because of this probe, an alpha cut can be performed.
- Notations
  - $v \in [m_i, M_i]$ which are the lower and upper bounds of v after the i probe.
  - $v_j \in [L_j, U_j]$ which are the lower and upper bounds of v_j.
- Formulas for Star2
  - $v_j \in [v_{min}, v_{j,1}]$ after T_j is probed.
  - After the ith probe, $v \in [m_0, M_{i-1} + Pr_i \times (v_{i,1} - v_{max})]$.
    - ▷ *m_i is unchanged, but $M_i = M_{i-1} + Pr_i \times (v_{i,1} - v_{max})]$*
    - ▷ *A_i is updated according to 11, but B_i is unchanged.*
  - In comparison, for Star1
    - ▷ $m_i = m_{i-1} + Pr_i \times (v_i - v_{min})]$
    - ▷ $M_i = M_{i-1} + Pr_i \times (v_i - v_{max})]$
    - ▷ *Both A_i and B_i are updated.*

Illustration: Star1 and Star2 probing



Star2: MIN node, general case

- p is a MIN chance node. Thus each child p_i is a MAX node.
- We have probed the first child of T_i and obtained $v_{i,1}$.
  - Since p_i is a MAX node, $v_{i,1}$ is a lower bound of v_i which is usually larger than the minimum possible value v_{min}.
  - The lower bound of v_i is thus raised.
  - It is possible because of this probe, a beta cut can be performed.
- Notations
  - $v \in [m_i, M_i]$ which are the lower and upper bounds of v after the i probe.
  - $v_j \in [L_j, U_j]$ which are the lower and upper bounds of v_j.
- Formulas for Star2
  - $v_j \in [v_{j,1}, v_{max}]$ after T_j is probed.
  - After the ith probe, $v \in [m_{i-1} + Pr_i \times (v_{i,1} - v_{min}), M_0$.
    - ▷ $m_i = m_{i-1} + Pr_i \times (v_{i,1} - v_{max})]$, ***but*** M_i ***is unchanged.***
    - ▷ A_i ***is unchanged, but*** B_i ***is updated according to 12.***
  - In comparison, for Star1
    - ▷ $m_i = m_{i-1} + Pr_i \times (v_i - v_{min})]$
    - ▷ $M_i = M_{i-1} + Pr_i \times (v_i - v_{max})]$
    - ▷ ***Both*** A_i ***and*** B_i ***are updated.***

Algorithm: Chance_Search with probing (1/2)

- **Algorithm** $F3.2'$**(position** p**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// max node**
  - **determine the successor positions** $p_1, \ldots, p_b$**;**
  - **if** $b = 0$ **// a terminal node**
    or $depth = 0$ **// remaining depth to search**
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - **then return** $f(p)$**; else begin**
    - ▷ $m := -\infty$;
    - ▷ ***for*** $i := 1$ ***to*** b ***do***
    - ▷ ***begin***
    - ▷ ***if*** p_i ***is to play a chance node*** x
      then $t := Star2\_F3.2'(p_i, x, \max\{alpha, m\}, beta, depth - 1)$;
    - ▷ ***else*** $t := G3.2'(p_i, \max\{alpha, m\}, beta, depth - 1)$;
    - ▷ ***if*** $t > m$ ***then*** $m := t$;
    - ▷ ***if*** $m \geq beta$ ***then return***(m); ***// beta cut off***
    - ▷ ***end;***
  - **end;**
  - **return** m**;**

Algorithm: Chance_Search with probing (2/2)

- **Algorithm** $G3.2'$**(position** p**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// min node**
  - **determine the successor positions** $p_1, \ldots, p_b$**;**
  - **if** $b = 0$ **// a terminal node**
    or $depth = 0$ **// remaining depth to search**
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - **then return** $f(p)$**; else begin**
    - ▷ $m := \infty$;
    - ▷ ***for*** $i := 1$ ***to*** b ***do***
    - ▷ ***begin***
    - ▷ ***if*** p_i ***is to play a chance node*** x
      then $t := Star2\_G3.2'(p_i, x, alpha, \min\{beta, m\}, depth - 1)$;
    - ▷ ***else*** $t := F3.2'(p_i, alpha, \min\{beta, m\}, depth - 1)$;
    - ▷ ***if*** $t < m$ ***then*** $m := t$;
    - ▷ ***if*** $m \leq alpha$ ***then return(***m***); // alpha cut off***
    - ▷ ***end;***
  - **end;**
  - **return** m**;**

Star2 (1/2)

- **Algorithm** $Star2\_F3.2'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - **// a max chance node x with c choices k_1, ..., k_c**
  - **// the ith choice happens with the probability Pr$_i$**
  - **determine the possible values of the chance node x to be $k_1, \ldots, k_c$**
  - **initialize A_0, B_0, m_0 and M_0 as in $Star1\_F3.1'$**
  - **// Do an exact probing for each choice to find cut off's.**
  - **for each choice i from 1 to c do**
    - ▷ ***Let p_i be the position obtained from p by making x the choice k_i.***
    - ▷ ***// do an exact probe on the first MIN child of p_i***
      v := $F3.2'(p_{i,1}$,**max**$\{A_{i-1}$,$v_{min}\}$,**min**$\{B_{i-1}$,$v_{max}\}$,$depth)$
    - ▷ ***update A_i and M_i as in $Star1\_F3.1'$***
    - ▷ ***If $M_i \leq alpha$ then return M_i; // alpha cut off***
  - **// normal exhaustive search phase**
    // no cut off is found in the above, do the normal Star1 search.
    // Chance node cut off I may happen.
    // Chance node cut off II may happen.
  - **return** $vexp = Star1\_F3.1(p, x, alpha, beta, depth)$;

Star2 (2/2)

- **Algorithm** $Star2\_G3.2'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** $depth$**)**
  - // a min chance node x with c choices k_1, ..., k_c
  - // the ith choice happens with the probability $\Pr_i$
  - determine the possible values of the chance node x to be $k_1, \ldots, k_c$
  - initialize A_0, B_0, m_0 and M_0 as in $Star1\_G3.1'$
  - // Do an exact probing for each choice to find cut off's.
  - for each choice i from 1 to c do
    - ▷ *Let* p_i *be the position obtained from* p *by making* x *the choice* k_i*.*
    - ▷ *// do an exact probe on the first MAX child of* p_i
      v := $G3.2'(p_{i,1}$,**max**$\{A_{i-1}$,$v_{min}\}$,**min**$\{B_{i-1}$,$v_{max}\}$,$depth)$
    - ▷ *update* B_i *and* m_i *as in* $Star1\_G3.1'$
    - ▷ *If* $m_i \geq beta$ *then return* m_i*; // beta cut off*
  - // normal exhaustive search phase
    // no cut off is found in the above, do the normal Star1 search.
    // Chance node cut off I may happen.
    // Chance node cut off II may happen.
  - return $vexp = Star1\_G3.1(p, x, alpha, beta, depth)$;

Comments for Star2

- During the exact probe phase, some bounds are known which can be used to update the search window.
- If no cut off is found in the probing phase, then we need to do the exhaustive searching phase.
  - The searched branches in the probing phase do not need to be researched again.

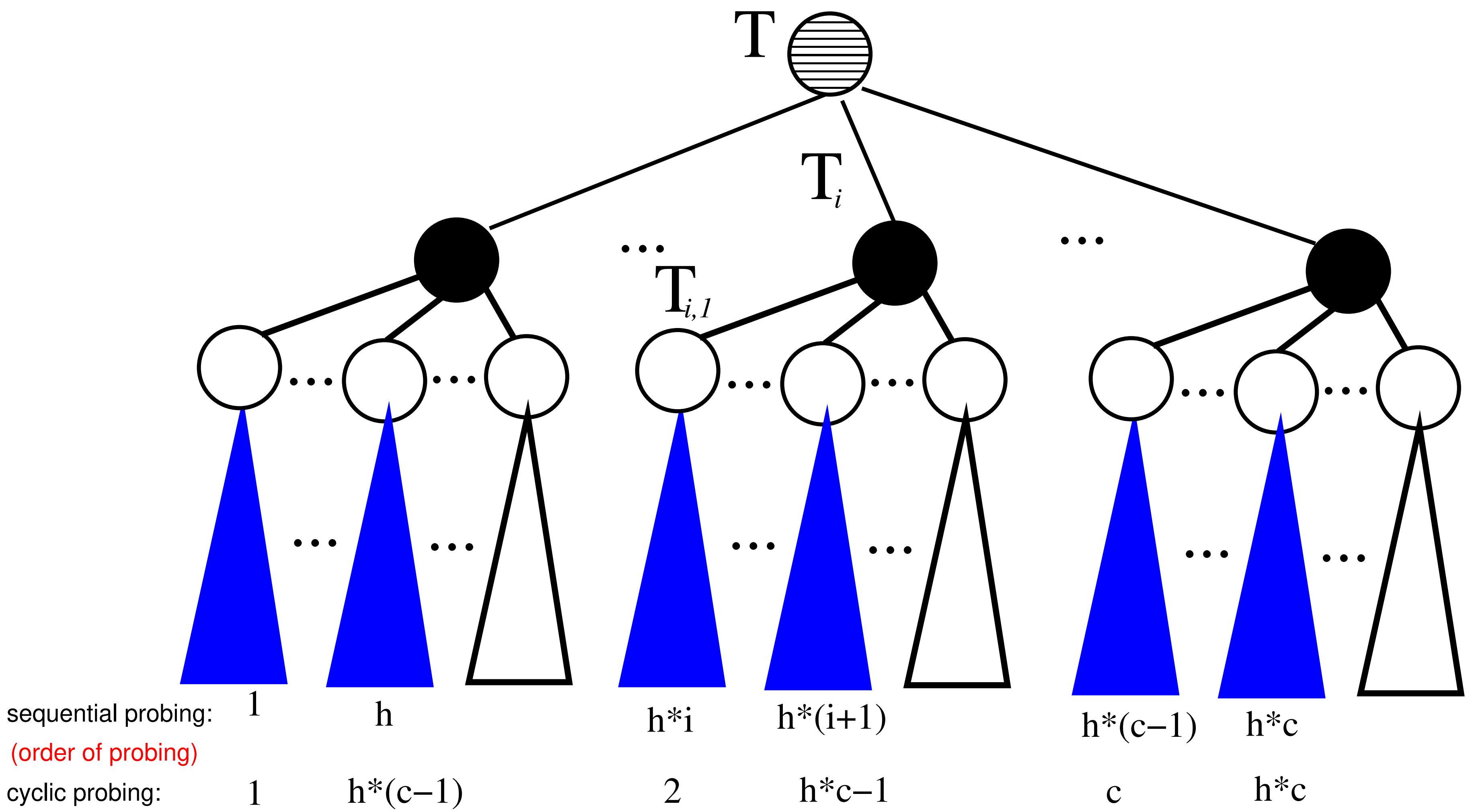
More ideas for probes

- **Move ordering in exploring the choices is critical in performance.**
- **Picking which child to do the probe is also critical.**
- **Can do exact probes on h children, called probing factor $h > 1$, of a choice instead of fixing the number of probings to be exactly one.**
- **May decide to probe different number of children for each choice.**

Probing orders

- **Two types of probing orders with a probing factor h**
  - **Cyclic probing**
    - ▷ ***Probe one child of a choice at one time for all choices, and do this for h rounds.***
    - ▷ ***for $j = 1$ to h do***
      for $i = 1$ to c do
      probe the jth child of the ith choice
  - **Sequential probing**
    - ▷ ***Probe h children of a choice at one time and then do it for each choice in sequence***
    - ▷ ***for $i = 1$ to c do***
      probe h children of the ith choice
    - ▷ ***Switch lines 6 and 7 in algorithms Star2.5_F3.2.5' and Star2.5_G3.2.5'.***
- **Special cases**
  - ▷ ***When $h = 0$, Star2 == Star1.***
  - ▷ ***When $h = 1$, cyclic probing == sequential probing and also Star2 == Star2.5.***

Illustration: Star2.5 probing



Star2.5: Probe the first h children of each Ti

Chance_Search with h cyclic probings (1/2)

- **Algorithm $F3.2.5'$(position p, value $alpha$, value $beta$, integer $depth$, integer h)**
  - **// max node**
  - **determine the successor positions $p_1, \ldots, p_b$;**
  - **if $b = 0$ // a terminal node**
    or $depth = 0$ // remaining depth to search
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - **then return $f(p)$; else begin**
    - ▷ *$m := -\infty$;*
    - ▷ ***for** $i := 1$ **to** b **do***
    - ▷ ***begin***
    - ▷   ***if** p_i **is to play a chance node** x*
          ***then** $t := Star2\_F3.2.5'(p_i, x, \max\{alpha, m\}, beta, depth - 1, h)$;*
    - ▷   ***else** $t := G3.2.5'(p_i, \max\{alpha, m\}, beta, depth - 1, \mathbf{h})$;*
    - ▷   ***if** $t > m$ **then** $m := t$;*
    - ▷   ***if** $m \geq beta$ **then return**(m); // beta cut off*
    - ▷ ***end;***
  - **end;**
  - **return m;**

Chance_Search with h cyclic probings (2/2)

- **Algorithm** $G3.2.5'$**(position** p**, value** $alpha$**, value** $beta$**, integer** $depth$**, integer** h**)**
  - **// min node**
  - **determine the successor positions** $p_1, \ldots, p_b$**;**
  - **if** $b = 0$ **// a terminal node**
    or $depth = 0$ **// remaining depth to search**
    or time is running up // from timing control
    or some other constraints are met // add knowledge here
  - **then return** $f(p)$**; else begin**
    - ▷ *$m := \infty$;*
    - ▷ ***for*** *$i := 1$* ***to*** *b* ***do***
    - ▷ ***begin***
    - ▷ ***if*** *p_i* ***is to play a chance node*** *x*
      then *$t := Star2\_G3.2.5'(p_i, x, alpha, \min\{beta, m\}, depth - 1, h)$;*
    - ▷ ***else*** *$t := F3.2.5'(p_i, alpha, \min\{beta, m\}, depth - 1, h)$;*
    - ▷ ***if*** *$t < m$* ***then*** *$m := t$;*
    - ▷ ***if*** *$m \leq alpha$* ***then return****(m); // alpha cut off*
    - ▷ ***end;***
  - **end;**
  - **return** m**;**

Star2.5: cyclic probing (1/2)

- **Algorithm** $Star2.5\_F3.2.5'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** h**) //** h **is the probing factor**
  - **// a MAX chance node** x **with** c **choices** k_1**, . . .,** k_c
  - **// the** i**th choice happens with the probability Pr**$_i$
  - **determine the possible values of the chance node** x **to be** $k_1, \ldots, k_c$
  - **initialize** A_0**,** B_0**,** m_0 **and** M_0 **as in** $Star1\_F3.1'$
  - **// Do a cyclic probing to decide whether some cut off can be performed.**
  - **6: for** j **from** 1 **to** h **do**
    7: for each choice i **from** 1 **to** c **do**
    - ▷ ***Let*** p_i ***be the position obtained from*** p ***by making*** x ***the choice*** k_i***.***
    - ▷ ***// do an exact probe on the*** j***th MIN child of*** p_i***.***
      v ***:=*** $G3.2'(p_{i,j}$***,max{***A_{i-1}***,***v_{min}***},min{***B_{i-1}***,***v_{max}***},***$depth)$
    - ▷ ***update*** A_i ***and*** M_i ***as in*** $Star1\_F3.1'$
    - ▷ ***If*** $M_i \leq alpha$ ***then return*** M_i***; // alpha cut off***
  - **// normal exhaustive search phase**
    // no cut off is found in the above, do the normal Star1 search.
    // Chance node cut off I may happen.
    // Chance node cut off II may happen.
  - **return** $vexp = Star1\_F3.1(p, x, alpha, beta, depth)$**;**

Star2.5: cyclic probing (2/2)

- **Algorithm** $Star2.5\_G3.2.5'$**(position** p**, node** x**, value** $alpha$**, value** $beta$**, integer** h**) //** h **is the probing factor**
  - **// a MIN chance node** x **with** c **choices** k_1**, ...,** k_c
  - **// the** i**th choice happens with the probability Pr**$_i$
  - **determine the possible values of the chance node** x **to be** $k_1, \ldots, k_c$
  - **initialize** A_0**,** B_0**,** m_0 **and** M_0 **as in** $Star1\_G3.1'$
  - **// Do a cyclic probing to decide whether some cut off can be performed.**
  - **6: for** j **from** 1 **to** h **do**
    7: for each choice i **from** 1 **to** c **do**
    - ▷ ***Let*** p_i ***be the position obtained from*** p ***by making*** x ***the choice*** k_i***.***
    - ▷ ***// do an exact probe on the*** j***th MAX child of*** p_i***.***
      v := $F3.2'(p_{i,j}$,**max**$\{A_{i-1}$,$v_{min}\}$,**min**$\{B_{i-1}$,$v_{max}\}$,$depth)$
    - ▷ ***update*** B_i ***and*** m_i ***as in*** $Star1\_G3.1'$
    - ▷ ***If*** $m_i \geq beta$ ***then return*** m_i***; // beta cut off***

    // normal exhaustive search phase
    // no cut off is found in the above, do the normal Star1 search.
    // Chance node cut off I that is similar to beta cut off may happen.
    // Chance node cut off II that is similar to alpha cut off may happen.
  - **return** $vexp = Star1\_G3.1(p, x, alpha, beta, depth)$**;**

Comments

- **Experimental results provided in [Ballard '83] on artificial game trees.**
  - **Star1 may not be able to cut more than 20% of the leaves.**
  - **Star2.5 with $h = 1$, i.e. Star2, cuts more than 59% of the nodes and is about twice better than Star1.**
  - **Sequential probing is best when $h = 3$ which cuts more than 65% of the nodes and roughly cut about the same nodes as Star2.5 using the same probing factor.**
  - **Sequential probing gets worse when $h > 4$. For example, it only cut 20% of the leaves when $h = 20$.**
  - **Star2.5 continues to cut more nodes when h gets larger, though the gain is not that great. At $h = 3$, about 70% of the nodes are cut. At $h = 20$, about 72% of the nodes are cut.**
- **Need to store the bounds and when the bounds produces cuts in the hash table for later to resume searching if needed later when the node is revisited.**
- **Better move ordering is also needed to get a fast cut off.**

Approximated Probes

- **We can also have heuristics for issuing approximated probes which returns approximated values.**
- **Strategy I: random probing of some promising choices**
  - **Do a move ordering heuristic to pick one or some promising choices to expand first.**
  - **These promising choices can improve the lower or upper bounds and can cause beta or alpha cut off.**
- **Strategy II: fast probing of all choices**
  - **Possible implementations**
    - ▷ *do a static evaluation on all choices*
    - ▷ *do a shallow alpha-beta searching on each choice*
    - ▷ *do a MCTS-like simulation on the choices*
  - **Use these information to decide whether you have enough confidence to do a cut off.**

Using MCTS with chance nodes (1/2)

- **Assume a chance node x has c choices $k_1, \ldots, k_c$ and the ith choice happens with the probability Pr_i**
- **Selection**
  - **If x is picked in the PV during selection, then a random coin tossing according to the probability distribution of the choices is needed to pick which choice to descent.**
    - ▷ ***It is better to even the number of simulations done on each choice.***
    - ▷ ***Use random sampling without replacement. When every one is picked once, then start another round of picking.***
- **Expansion**
  - **If the last node in the PV is x, then expand all choices and simulate each choice some number of times.**
    - ▷ ***Watch out the discuss on maxing chance nodes in a searching path such as whether it is desirable to have 2 chance nodes in sequence ... etc.***

Using MCTS with chance nodes (2/2)

- **Simulation**
  - **When a chance node is to be simulated, then be sure to randomly, according to the probability distribution, pick a choice.**
    - *Use some techniques to make sure you are doing an effective sampling when the number of choices is huge*
    - *Watch out what are "reasonable" in a simulated plyout on the mixing of chance nodes.*
- **Back propagation**
  - **The UCB score of x is $w_i + c\sqrt{(lnN/N_i)}$ where w_i is the weighted winning rate, or score, of the children, N_i is the total number of simulations done on all choices. and N is the total number of simulations done on the parent of x.**

Sparse sampling (1/2)

- **Assume in searching the number of possible outcomes in a, maybe chance, node is too large. A technique called sparse sampling can be used [Kearn et al 2002].**
  - **Can also be used in the expansion phase of MCTS.**
- **Ideas:**
  - **The number of choices, $a = |\mathcal{A}|$, considered is enlarged as the number of visits to the node increases.**
  - **Use the current choice set as an estimation of its goodness.**
  - **Only consider k_t randomly selected choices, called $\mathcal{S}_t$, in the first t visits where $k_t = \lceil c * t^\alpha \rceil$, and c and α are constants.**
- **Algorithm SS for sparse sampling**
  - $t := 1$
  - **Initial k_t to be a small constant, say 1.**
  - **Initial the candidate set $\mathcal{S}$ to be an empty set.**
  - **Randomly pick k_t children from $\mathcal{A}$ into $\mathcal{S}$**
  - **loop: Performs some t' samplings from $\mathcal{S}$.**
    - ▷ *Add randomly $k_{t+t'} - k_t$ new children from $\mathcal{A}$ into $\mathcal{S}$*
    - ▷ $t \mathrel{+}= t'$
  - **goto loop**

Sparse sampling (2/2)

- **The estimated value is accurate with a high probability [Kearns et al 2002] [Lanctot et al 2013]**
- **Theorem:**

$$Pr(|\tilde{V} - V| \leq \lambda \cdot d) \geq 1 - (2 \cdot k_t \cdot c)^d exp\{\frac{-\lambda^2 \cdot k_t}{2 \cdot v_{max}^2}\},$$

where

- ▷ k_t ***is the number of choices considered with*** t ***samplings,***
- ▷ $\tilde{V}$ ***is the estimation considering only*** k_t ***choices,***
- ▷ V ***is the value considering all choices,***
- ▷ c ***is the actual number of choices,***
- ▷ d ***is the depth simulated,***
- ▷ $\lambda \in (0, 2 \cdot v_{max}]$ ***is a parameter chosen, and***
- ▷ v_{max} ***is the maximum possible value.***

- **Note: the proof is done by making sampling with replacement, while the algorithm asks for sampling without replacement.**

Comments

- **Chance node introduces a large searching space that needs careful treatment.**
  - **Need information in every possible branch to come out with a good strategy.**
- **Suppose that in each move,**
  - **on**
    - ▷ ***a prior chance node: you have m possible moves followed by r different random outcomes.***
    - ▷ ***a posteriori chance node: there are r different random outcomes from the coin toss and m possible moves followed.***
  - **Depending on r and m, good search algorithms can be designed.**
    - ▷ ***When $m >> r$, you may plainly enumerate all r alternatives.***
    - ▷ ***When $m << r$, may need to devise good strategies.***
- **Instead of looking for something that is sure-not-to-loss, may want something that is have-a-chance-to-win.**

References and further readings (1/2)

* Bruce W. Ballard The *-minimax search procedure for trees containing chance nodes Artificial Intelligence, Volume 21, Issue 3, September 1983, Pages 327-350
- Marc Lanctot, Abdallah Saffidine, Joel Veness, Chris Archibald, Mark H. M. Winands Monte-Carlo *-MiniMax Search Proceedings IJCAI, pages 580–586, 2013.
- 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.
- Lorentz, R.J. (2012). An MCTS Program to Play EinStein Würfelt Nicht!. In: van den Herik, H.J., Plaat, A. (eds) Advances in Computer Games. ACG 2011. Lecture Notes in Computer Science, vol 7168. Springer, Berlin, Heidelberg.

References and further readings (2/2)

- Jouandeau, N., Cazenave, T. (2014). Monte-Carlo Tree Reductions for Stochastic Games. In: Cheng, SM., Day, MY. (eds) Technologies and Applications of Artificial Intelligence. TAAI 2014. Lecture Notes in Computer Science(), vol 8916. Springer, Cham.
- S. Yen, C. Chou, J. Chen, I. Wu and K. Kao, ”Design and Implementation of Chinese Dark Chess Programs,” in IEEE Transactions on Computational Intelligence and AI in Games, vol. 7, no. 1, pp. 66-74, 2014.