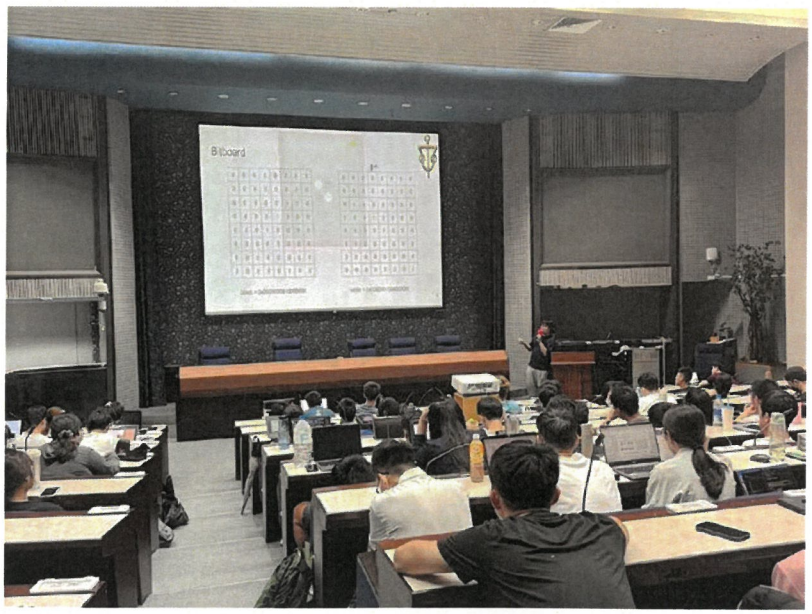
邀請中華民國斐陶斐榮譽學會榮譽會員演講成果報告

主辦單位	國立臺灣師範大學資訊工程學系
時　　間	112 年 9 月 6 日（三）14:20～16:00
地　　點	臺師大公館校區綜合館 3 樓國際會議廳
講　　題	如何打造一支厲害的黑白棋程式
講　　者	陳彥吉
主 持 人	葉梅珍教授

演講摘要

介紹講者在國內外比賽多次榮獲金牌的黑白棋程式的實作細節，其中包含了：走步產生、估值函數的設計、搜尋算法的改良、平行化遊戲樹搜索時的負載平衡、開局資料庫的設計。

活動照片（請呈現「中華民國斐陶斐榮譽學會」字樣或 Logo）



團體會員學校校長簽章：校長吳正己

年　　月　　日

如何打造一支厲害的黑白棋程式

講者:陳彥吉

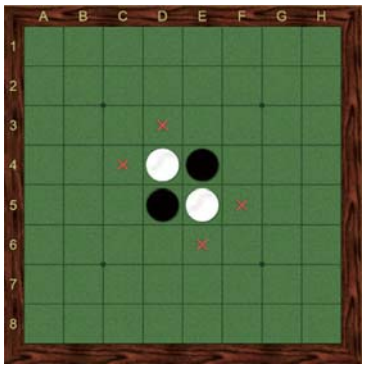
陳彥吉

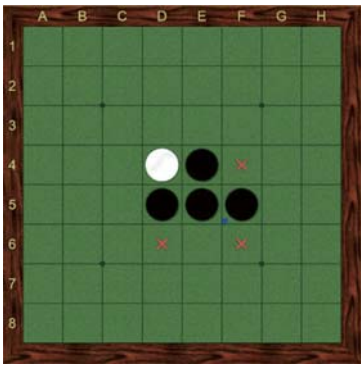
- 學歷
  - 國立臺灣師範大學數學系＋資訊工程系雙主修
  - 國立臺灣師範大學資訊工程所
- 就業？
  - 師大資工研究助理
  - 中研院研究助理
- 今天分享的內容
  - 跟畢業論文無關
  - 我大三至研究所畢業還做了 啥？
  - 黑白棋戰績: 16金6銀2銅

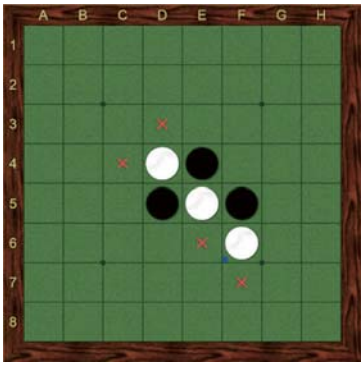
Outline

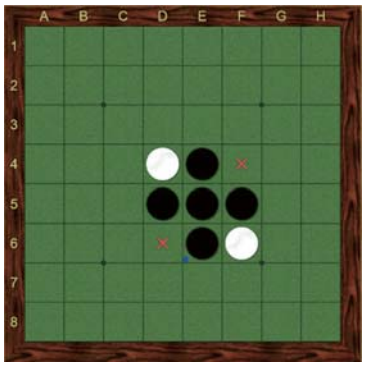
1. Introduction
2. Action generator
3. Evaluation function
4. Search algorithm
5. Parallel search algorithm
6. Opening book
7. Conclusion

Rule

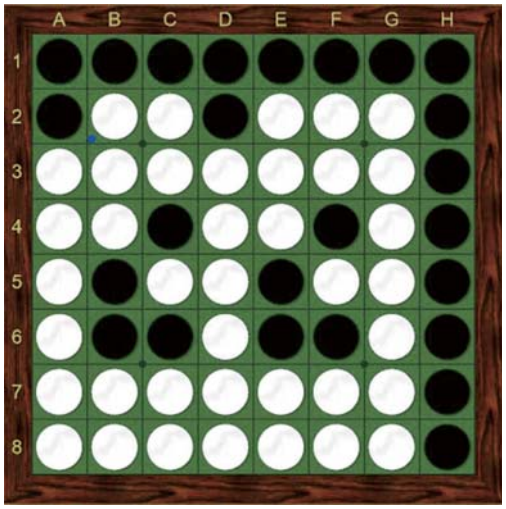




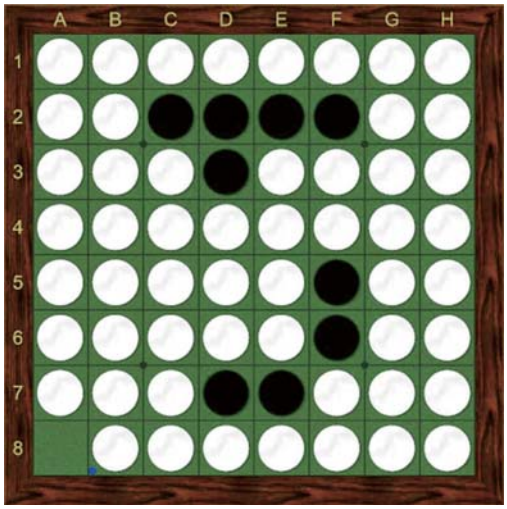




Rule



25-**39**



9-**54**

Maverick

- My Othello AI program, Maverick
  - Based on AlphaBeta search
  - Perfect end game search for final 24 layers
  - Opening book
- There is a trade-off between speed and accuracy.

	AlphaBeta	MCTS	AlphaZero
Speed (nodes/s)	10,000,000	100,000	1,000
Accuracy	Low ⟷ High		

Outline

1. Introduction
2. Action generator
3. Evaluation function
4. Search algorithm
5. Parallel search algorithm
6. Opening book
7. Conclusion

Bitboard

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

black = 0x0000000810000000

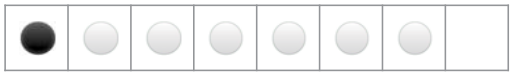
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

white = 0x0000001008000000

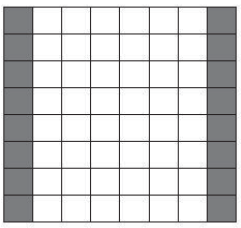
Sequential

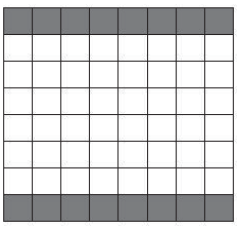
```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```

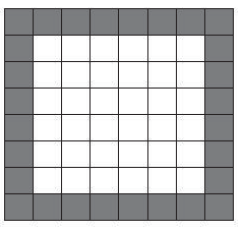

Sequential



```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```



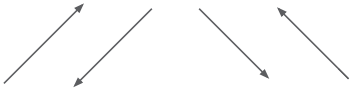




0x7e7e7e7e7e7e7e7e

0x00ffffffffffff00

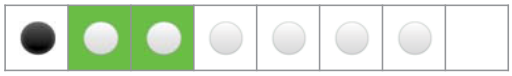
0x007e7e7e7e7e7e00



Sequential

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```


Sequential



```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```

Sequential

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```


Sequential

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```

Sequential

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```


Sequential

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```

Sequential

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```


Sequential

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```

x8 directions

```
+9   +8   +7
  \  ^  /
+1 <  0  > -1
  /  v  \
-7   -8   -9
```

Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```


Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```


Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```


Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```


Kogge-Stone

?

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```


Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```


Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```


Kogge-Stone

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

x8 directions

Hybrid

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
m =  (m >> 1) & m
t |= (t >> 2) & m
t |= (t >> 2) & m
actions |= (t >> 1) & empty
```

x8 directions

Hybrid

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
m =  (m >> 1) & m
t |= (t >> 2) & m
t |= (t >> 2) & m
actions |= (t >> 1) & empty
```

Hybrid

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
m =  (m >> 1) & m
t |= (t >> 2) & m
t |= (t >> 2) & m
actions |= (t >> 1) & empty
```


Hybrid

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
m =  (m >> 1) & m
t |= (t >> 2) & m
t |= (t >> 2) & m
actions |= (t >> 1) & empty
```

Hybrid

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
m =  (m >> 1) & m
t |= (t >> 2) & m
t |= (t >> 2) & m
actions |= (t >> 1) & empty
```


Hybrid

```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
m =  (m >> 1) & m
t |= (t >> 2) & m
t |= (t >> 2) & m
actions |= (t >> 1) & empty
```

Comparison

<Sequential>
```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
t |= (t >> 1) & m
actions |= (t >> 1) & empty
```

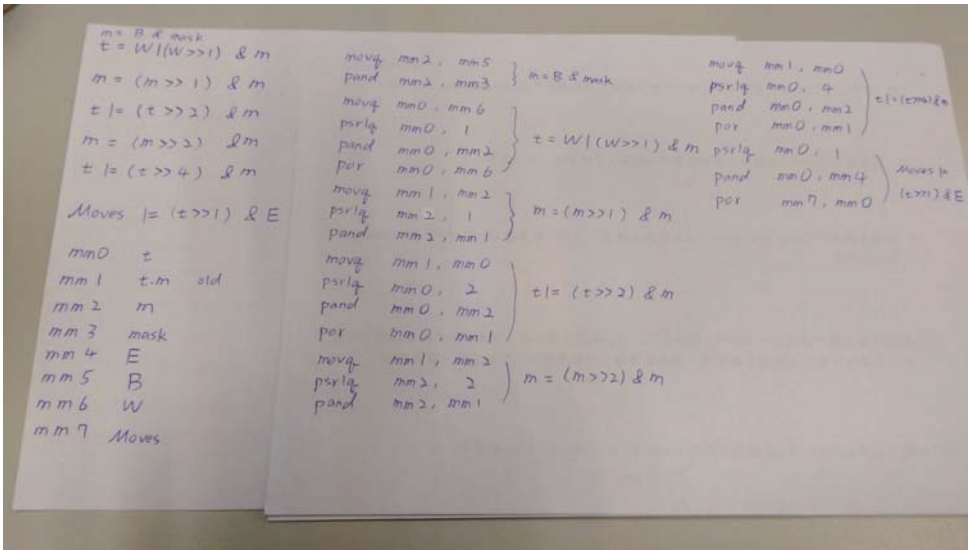
<Hybrid>
```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = (black >> 1) & m
t |= (t >> 1) & m
m =  (m >> 1) & m
t |= (t >> 2) & m
t |= (t >> 2) & m
actions |= (t >> 1) & empty
```

<Kogge-Stone>
```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
empty = ~(black | white)
t = black | ((black >> 1) & m)
m =  (m >> 1) & m
t |= (t >> 2) & m
m =  (m >> 2) & m
t |= (t >> 4) & m
actions |= ((t & white) >> 1) & empty
```

	Operator			Extra variables
	&	|	>>	
Sequential	7	6	7	t
Hybrid	6	4	6	t, m
Kogge-Stone	7	4	6	t, m

Assembly: MMX

- Generate actions in CPU without cache



Assembly: AVX2

組合語言的用書的問題

me
to [redacted]

[redacted]老師你好：
首先得謝謝老師願意開放名額來讓我修課。上個學期在林老師開的人工智慧課程中，為了加速搜尋演算速度，以及為了減少走步時反覆從記憶體要值次數，而研究了MMX指令。
最近，在與實驗室的學長們交流當中得知了AVX的存在。但是一直無法在intel網站上找到其規格書，後來改往尋找教學書(如附檔)。想詢問[redacted]老師，這本書適合學習AVX嗎?

...

bok%3A978-1-4842-0064-3.pdf
PDF

[redacted]同學:

我認為你想學習的東西，在我的組合語言課，都學不到。

我的課連浮點運算都不會上到，更別提MMX和SSE。

你說的AVX，我還是第一次聽到。

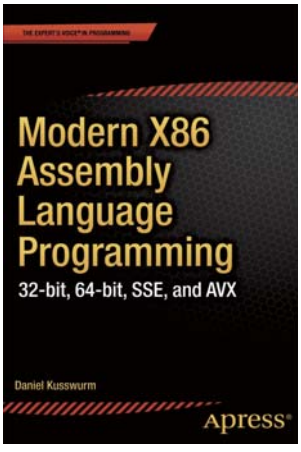
我自己的應用是在硬體的控制，用的都是基本的指令，浮點運算都用不到。

因此，除了因在計算機結構課程中有簡單的認識外，我可以說不了解。

我們的同學對於基本的組合語言指令都無法搞清楚，那些高級的指令實在沒有時間教到。

你若已經達到這個程度，我覺得來上我的組合語言課只是浪費時間而已。

Learning…



685 pages

3603 pages

Kogge-Stone with AVX2

Compute 4 directions at the same time

What is AVX2?

Advanced Vector Extensions (AVX)

AVX are extensions to the x86 instruction set architecture for microprocessors from Intel and AMD.

For example,

11 + 43 = 54

63 + 36 = 99

1163 + 4336 = 5499

Kogge-Stone with AVX2

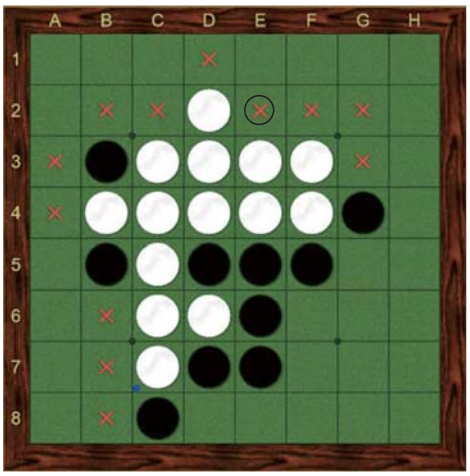
Compute 4 directions at the same time

```
mask = (0x007e7e7e7e7e7e00, 0x007e7e7e7e7e7e00,
        0x00ffffffffffff00, 0x7e7e7e7e7e7e7e7e)
shift1 = ( 9,  7,  8, 1)
shift2 = (18, 14, 16, 2)
shift4 = (36, 28, 32, 4)
empty = ~(black | white)
black = (black, black, black, black)
white = (white, white, white, white)
m = white & mask
t = black | ((black >> shift1) & m)
m =  (m >> shift1) & m
t |= (t >> shift2) & m
m =  (m >> shift2) & m
t |= (t >> shift4) & m
actions |= ((t & white) >> shift1) & empty
```

x2 for << and >>

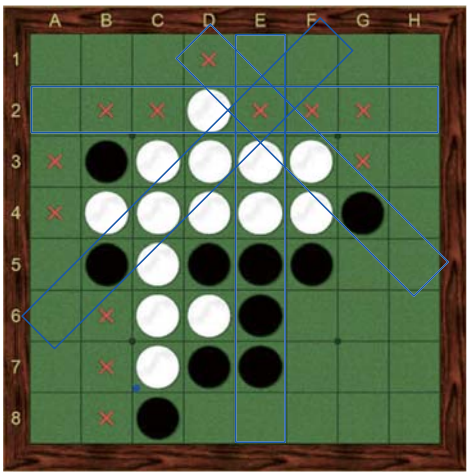
Reverse

Other othello programs



Reverse

Other othello programs



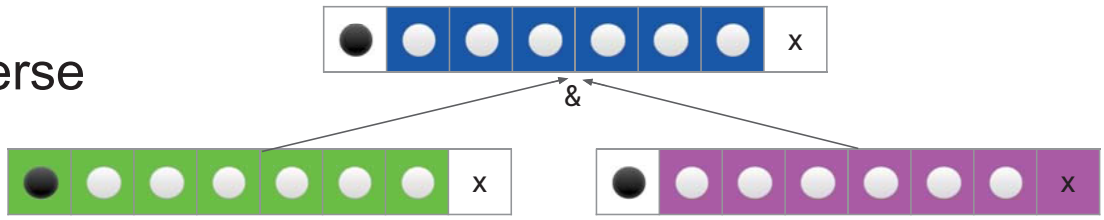
Look up a table: 64 * 4 * 3^8

Reverse


```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
t = black | ((black >> 1) & m)
mr = (mr>> 1) & mr
t |= (t >> 2) & mr
mr = (mr>> 2) & mr
t |= (t >> 4) & mr
reverse |= t & a
```

```
a = action | ((action << 1) & m)
ml = (ml<< 1) & ml
a |= (a << 2) & ml
ml = (ml<< 2) & ml
a |= (a << 4) & ml
```

Reverse




```
mask = 0x7e7e7e7e7e7e7e7e
m = white & mask
t = black | ((black >> 1) & m)
mr = (mr>> 1) & mr
t |= (t >> 2) & mr
mr = (mr>> 2) & mr
t |= (t >> 4) & mr
reverse |= t & a
```

```
a = action | ((action << 1) & m)
ml = (ml<< 1) & ml
a |= (a << 2) & ml
ml = (ml<< 2) & ml
a |= (a << 4) & ml
```


Speed test

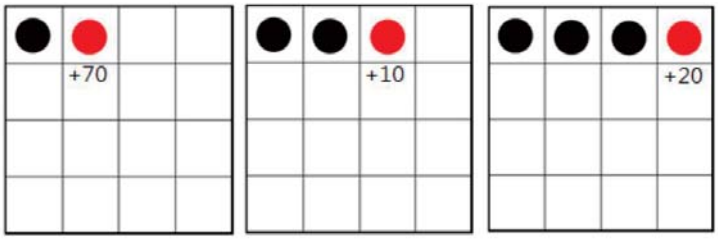
Method		**Speed** (nodes/s)
WZebra、Edax、Logistello		6,000,000 ~ 20,000,000
Normal	Sequential	58,783,372
	Hybrid	71,008,942
	Kogge-Stone	74,270,662
AVX2	Sequential	107,426,824
	Hybrid	130,786,882
	Kogge-Stone	139,810,153

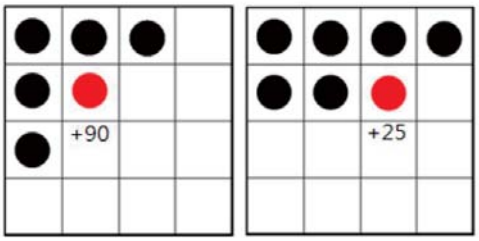
Outline

1. Introduction
2. Action generator
3. Evaluation function
4. Search algorithm
5. Parallel search algorithm
6. Opening book
7. Conclusion

Position value + stability

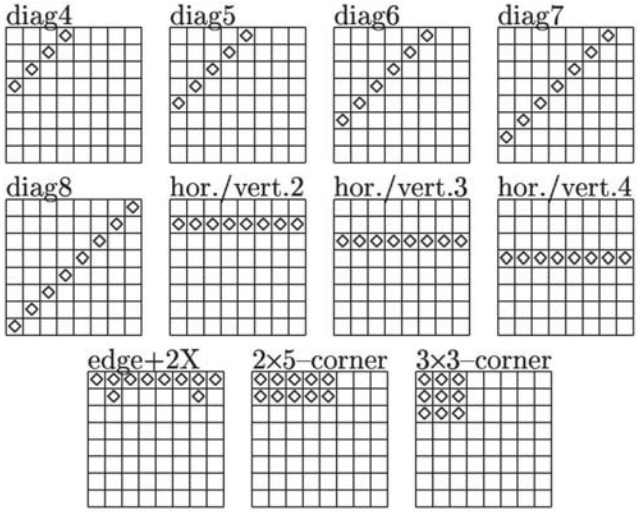
100	C -50	10	0				
	X -70	-5	-10				
		-10	-5				
			0				



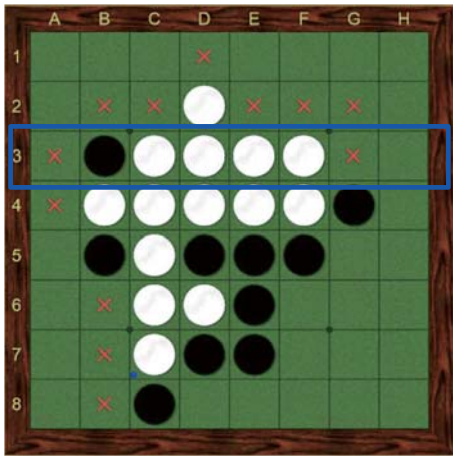


General Linear Evaluation Model (GLEM)

A pattern-based model



GLEM



$$(01222200)_3 = 1449$$

$$f_{p,i}(s) = \begin{cases} 1, & \text{if } s \text{ matches } p, i \text{ pattern} \\ 0, & otherwise \end{cases}$$

e.g. $f_{l3,1449}(s) = 1$

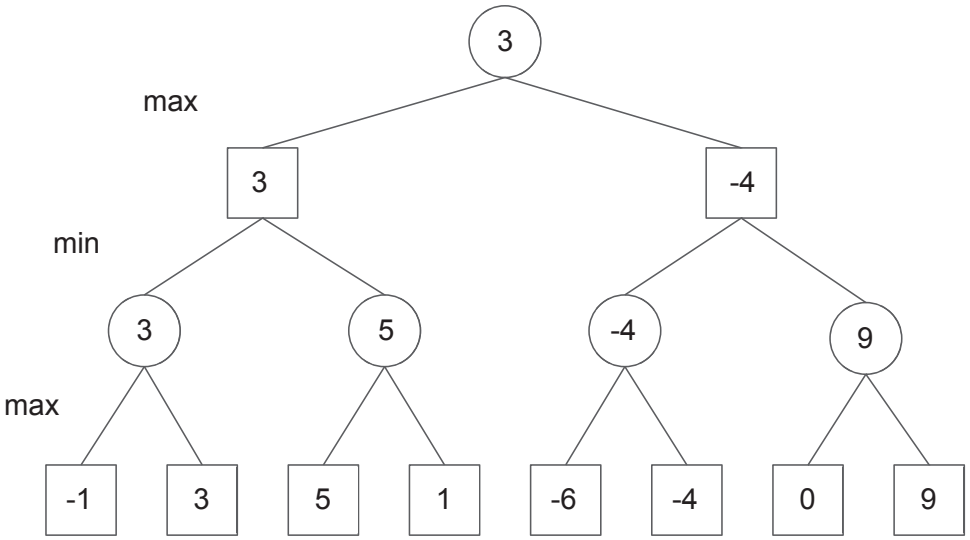
$$v(s) = \sum_{p \in P} \sum_{i=0}^{N_p} w_{p,i} f_{p,i}(s)$$

TD Learning: $v(s) = v(s) + \alpha\left(r + \gamma v(s') - v(s)\right)$

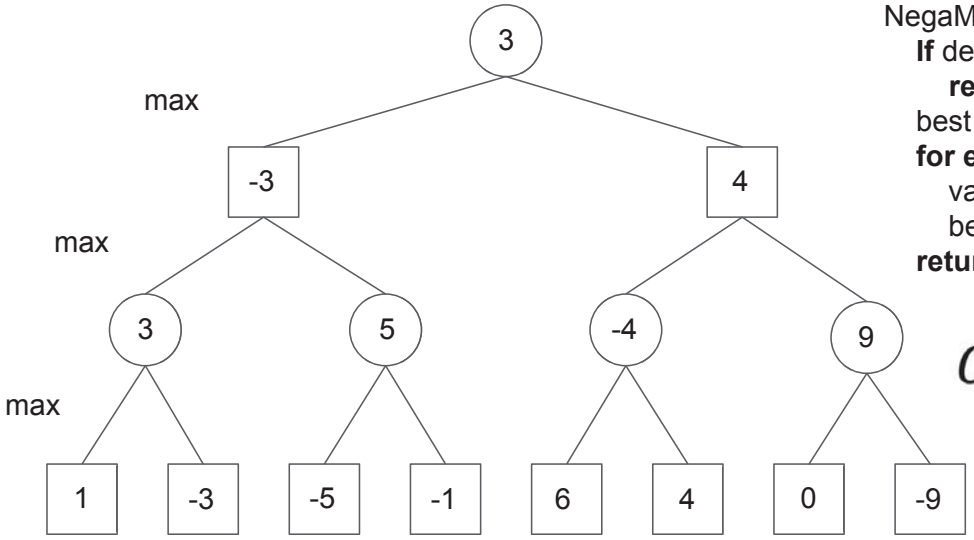
Outline

1. Introduction
2. Action generator
3. Evaluation function
4. Search algorithm
5. Parallel search algorithm
6. Opening book
7. Conclusion

Minimax



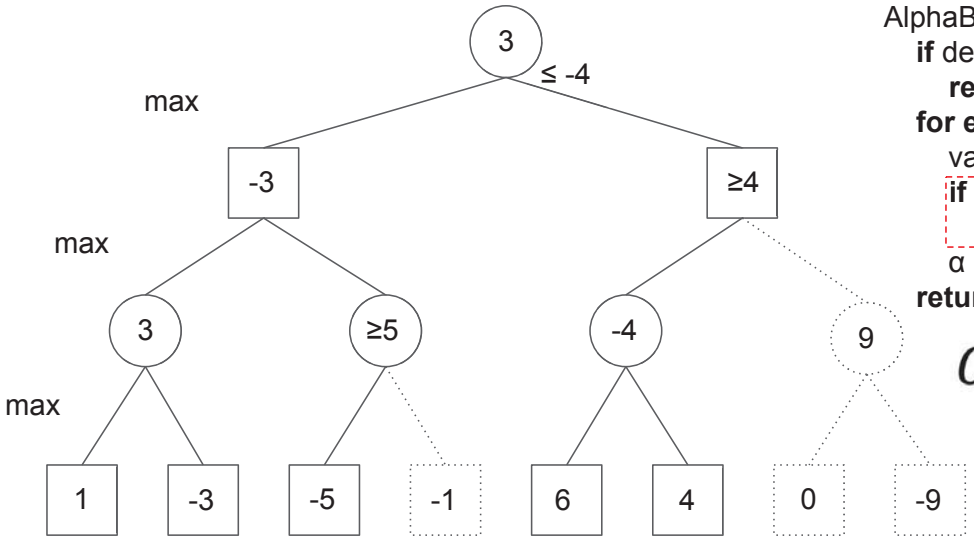
NegaMax




```
NegaMax(state, depth)
    If depth == 0 then
        return evaluate(state)
    best = -∞
    for each child of state
        value = -NegaMax(child, depth-1)
        best = max(best, value)
    return best
```

$$O(b^d)$$

Alpha-Beta pruning




```
AlphaBeta(state, α, β, depth)
    if depth == 0 then
        return evaluate(state)
    for each child of state
        value = -AlphaBeta(child, -β, -α, depth-1)
        if value >= β then
            return value
        α = max(α, value)
    return α
```

$$O(b^d) \rightarrow O(b^{d/2})$$

Move ordering

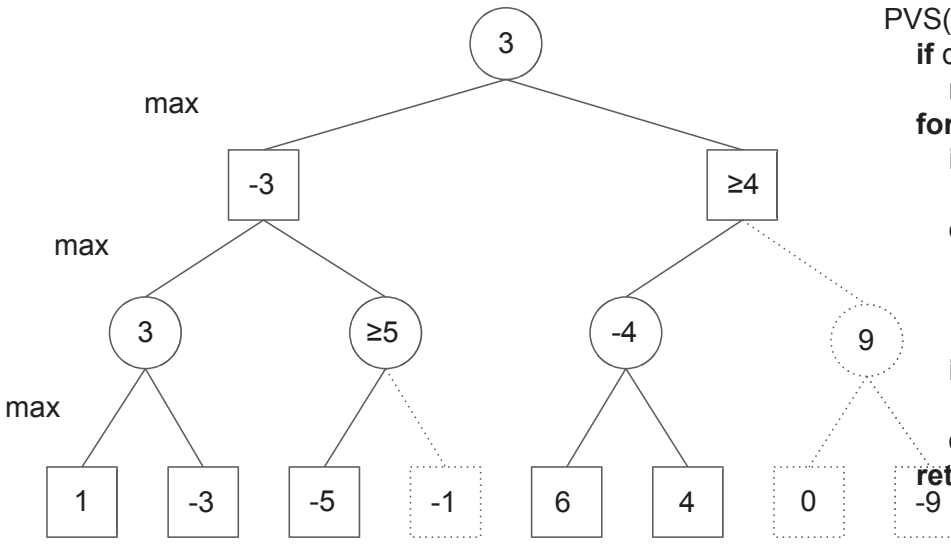
1. Ordering in advance

```
[A1, A8, H1, H8,
 C1, F1, H3, H6, F8, C8, A6, A3,
 C3, F3, F6, C6,
 D1, E1, H4, H5, E8, D8, A5, A4,
 D3, E3, F4, F5, E6, D6, C5, C4,
 D2, E2, G4, G5, E7, D7, B5, B4,
 C2, F2, G3, G6, F7, C7, B6, B3,
 B1, G1, H2, H7, G8, B8, A7, A2,
 B2, G2, G7, B7]
```

2. Previous search result

	A	B	C	D	E	F	G	H
1	1	8	2	4				
2		9	7	6				
3			3	5				
4								
5								
6								
7								
8								

Principal Variation Search (like NegaScout)




```
PVS(state, α, β, depth)
  if depth == 0 then
    return evaluate(state)
  for each child of state
    if child is first child then
      value = -PVS(child, -β, -α, depth-1)
    else                                  null window
      value = -PVS(child, -α-1, -α, depth-1)
      if α < value < β then
        value = -PVS(child, -β, -α, depth-1)
    if value >= β then
      return value
    α = max(α, value)
  return α
```

Narrow window

Score

- #winner - #loser

  Final score: -64, -63, …, -2, -1, 0, 1, 2, …, 63, 64 (integer)

  [α, β] : [-64, 64]

  [α, β] : [-1, 1] get 1 → [0, 2] or [1, 3]?

- #winner + #empty - #loser

  Final score: -64, -62, …, -4, -2, 0, 2, 4, …, 62, 64 (even number)

  [α, β] : [-1, 1] get 1 → [1, 3]...

Narrow window

Score

- #winner - #loser

  Final score: -64, -63, …, -2, -1, 0, 1, 2, …, 63, 64 (integer)

  ~~[α, β] : [-64, 64]~~

  [α, β] : [-1, 1] get 1 → [0, 2] or [1, 3]?

- #winner + #empty - #loser

  Final score: -64, -62, …, -4, -2, 0, 2, 4, …, 62, 64 (even number)

  [α, β] : [-1, 1] get 1 → [1, 3]...

Narrow window

Score

- #winner - #loser

  Final score: -64, -63, …, -2, -1, 0, 1, 2, …, 63, 64 (integer)

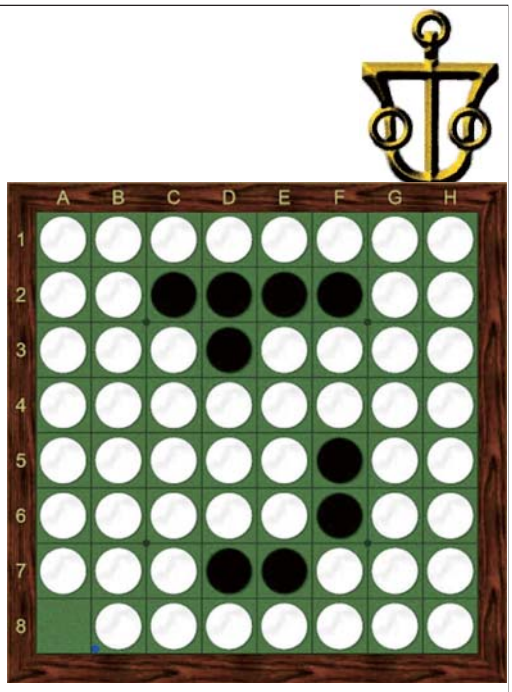
  ~~[α, β] : [-64, 64]~~

  [α, β] : [-1, 1] get 1 → [0, 2] or [1, 3]?

- #winner + #empty - #loser

  Final score: -64, -62, …, -4, -2, 0, 2, 4, …, 62, 64 (even number)

  [α, β] : [-1, 1] get 1 → [1, 3]...

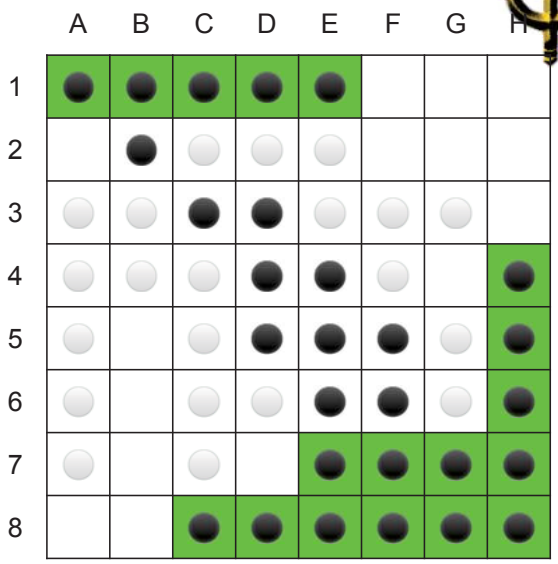


9-**54**

Stability

stability = #my_stability - (64 - #my_stability)

```
PVS(state, α, β, depth)
    if depth == 0 then
        return evaluate(state)
    stable_value = stability(state)
    if stable_value >= β then
        return stable_value
    for each child of state
        if child is first child then
            value = -PVS(child, -β, -α, depth-1)
        else
            value = -PVS(child, -α-1, -α, depth-1)
            if α < value < β then
                value = -PVS(child, -β, -α, depth-1)
        if value >= β then
            return value
        α = max(α, value)
    return α
```

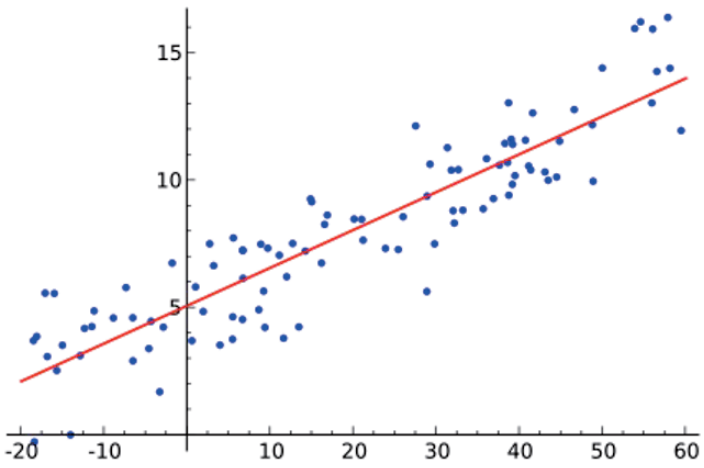


18 - 46 = -28

ProbCut

Just use a little math…

Linear Regression



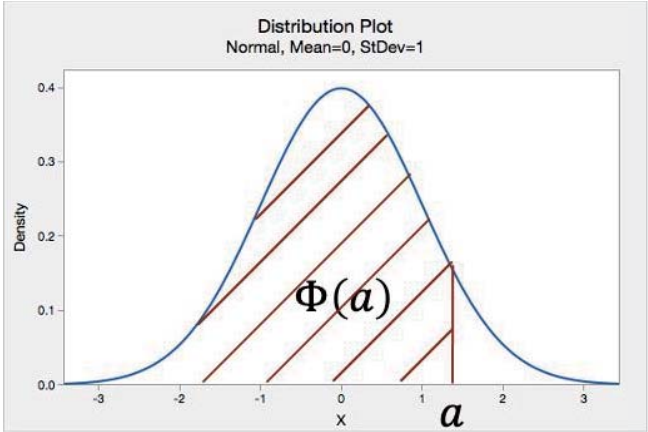
Normal distribution

PDF

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

CDF

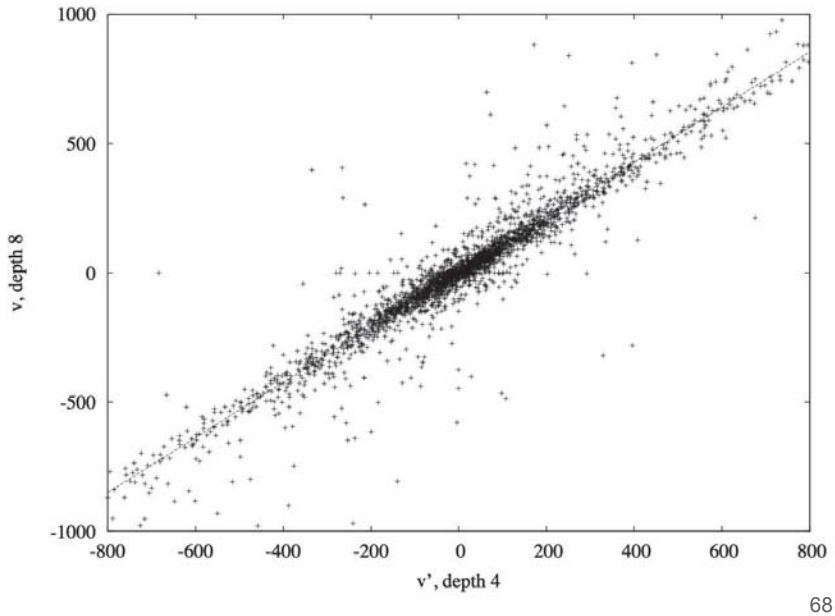
$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2} dt$$



Linear Regression with game

$$v = av' + b + e$$

$$e \sim \mathcal{N}(0, \sigma^2)$$
$$depth(v) > depth(v')$$



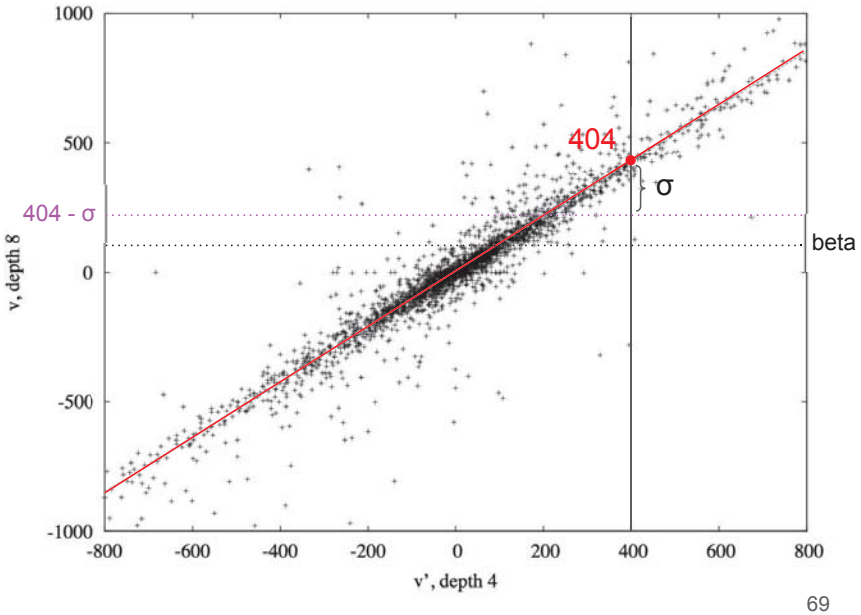
Beta cut

$$v = av' + b + e$$

$$e \sim \mathcal{N}(0, \sigma^2)$$
$$depth(v) > depth(v')$$

If we have
- v' = 400
- a = 1.01, b = 0, σ = 200
- beta = 100

Then v = 404 + e



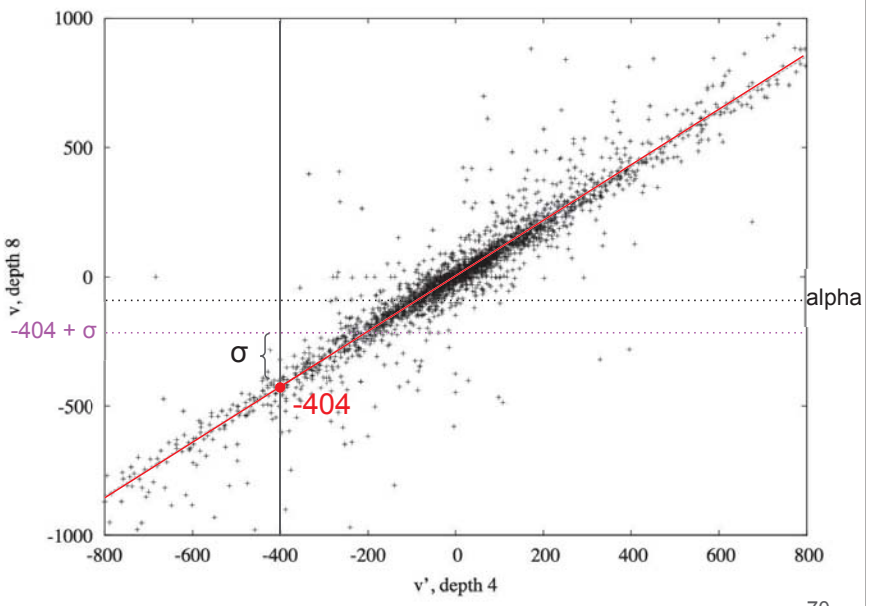
Alpha cut

$$v = av' + b + e$$

$$e \sim \mathcal{N}(0, \sigma^2)$$
$$depth(v) > depth(v')$$

If we have
- v' = -400
- a = 1.01, b = 0, σ = 200
- alpha = -100

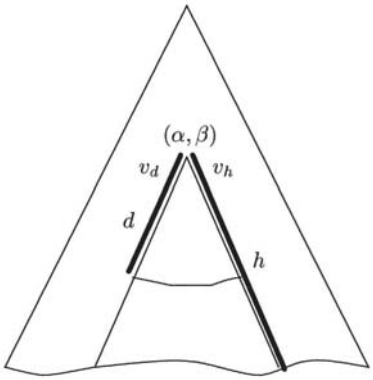
Then v = -404 + e



ProbCut

- $v_h \geq \beta$ // beta cut
- $\Leftrightarrow a \times v_d + b + e \geq \beta$
- $\Leftrightarrow (a \times v_d + b - \beta)/\sigma \geq -e/\sigma$
- $\Leftrightarrow (a \times v_d + b - \beta)/\sigma \geq \Phi^{-1}(p)$
- $\Leftrightarrow v_d \geq (\Phi^{-1}(p) \times \sigma + \beta - b)/a$

- $v_h \leq \alpha$ // alpha cut
- $\Leftrightarrow v_d \leq (-\Phi^{-1}(p) \times \sigma + \alpha - b)/a$



ProbCut pseudocode

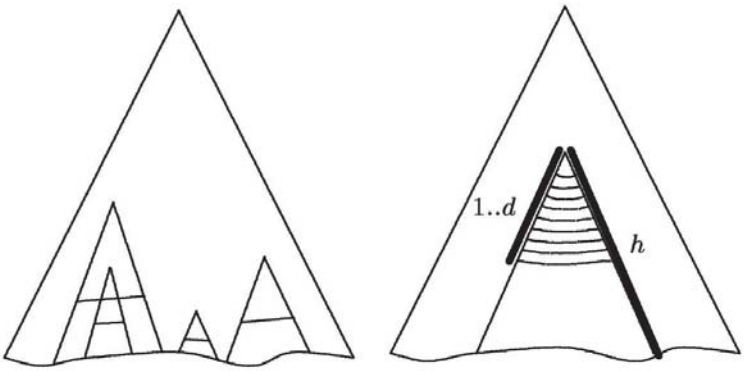
```
int PC(int α, int β, int depth) {
    const float T = 1.5;
    const int DP = 4;
    const int D = 8;

    if (depth == 0) return evaluate();
    if (depth == D) {
        int bound;
        bound = round((T*σ+β-b)/a);
        if (PC(bound-1, bound, DP) >= bound)
            return β;
        bound = round((-T*σ+α-b)/a);
        if (PC(bound, bound+1, DP) <= bound)
            return α;
    }
    /* and then alpha-beta search */
    ...
}
```

Multi-ProbCut

Let's play ProbCut anywhere!

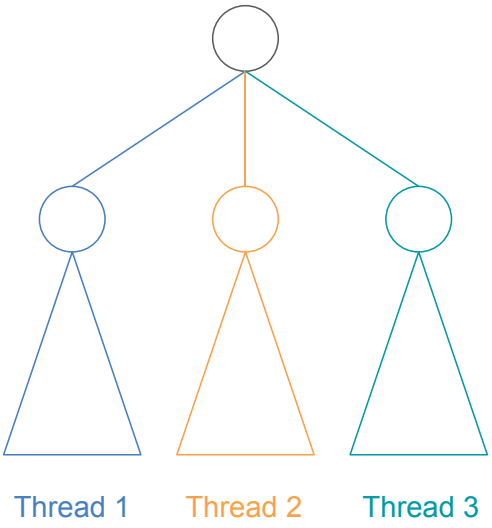
- Situation (occupy the corner?)
- Progression
- ProbCut depth



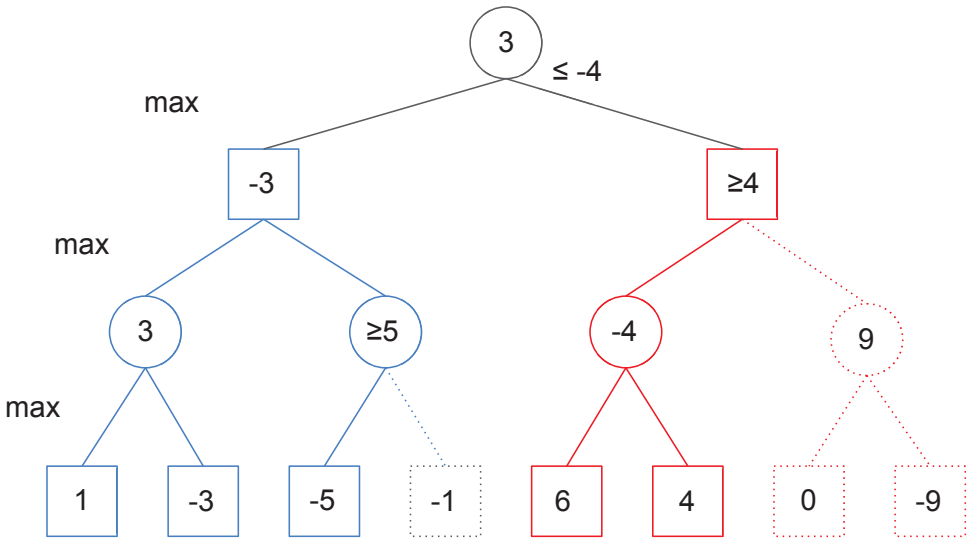
Outline

1. Introduction
2. Action generator
3. Evaluation function
4. Search algorithm
5. Parallel search algorithm
6. Opening book
7. Conclusion

Root parallel

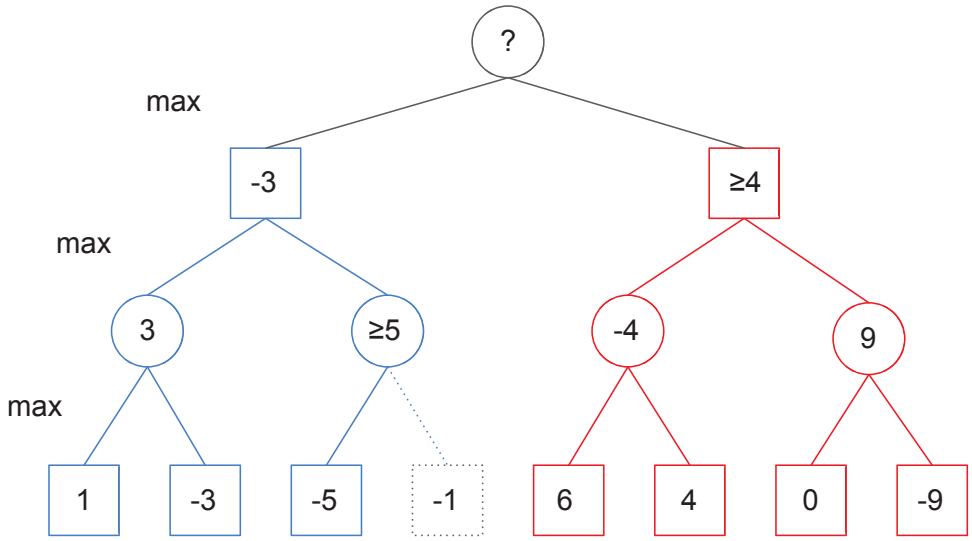


Root parallel issue



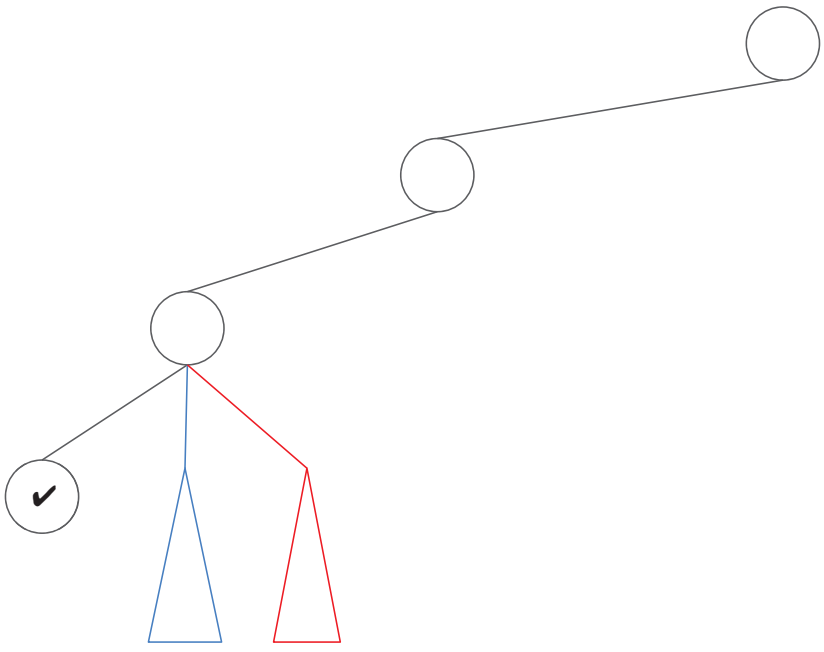
$$O\left(b^{d/2}\right) \rightarrow O\left(b^{d}\right)/n$$

Root parallel issue

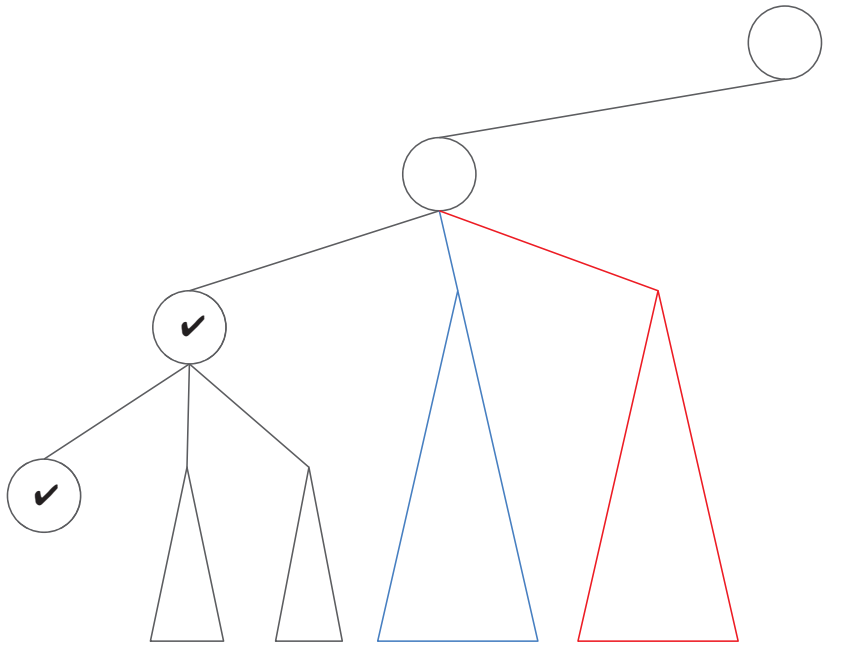


$$O(b^{d/2}) \rightarrow O(b^d)/n$$

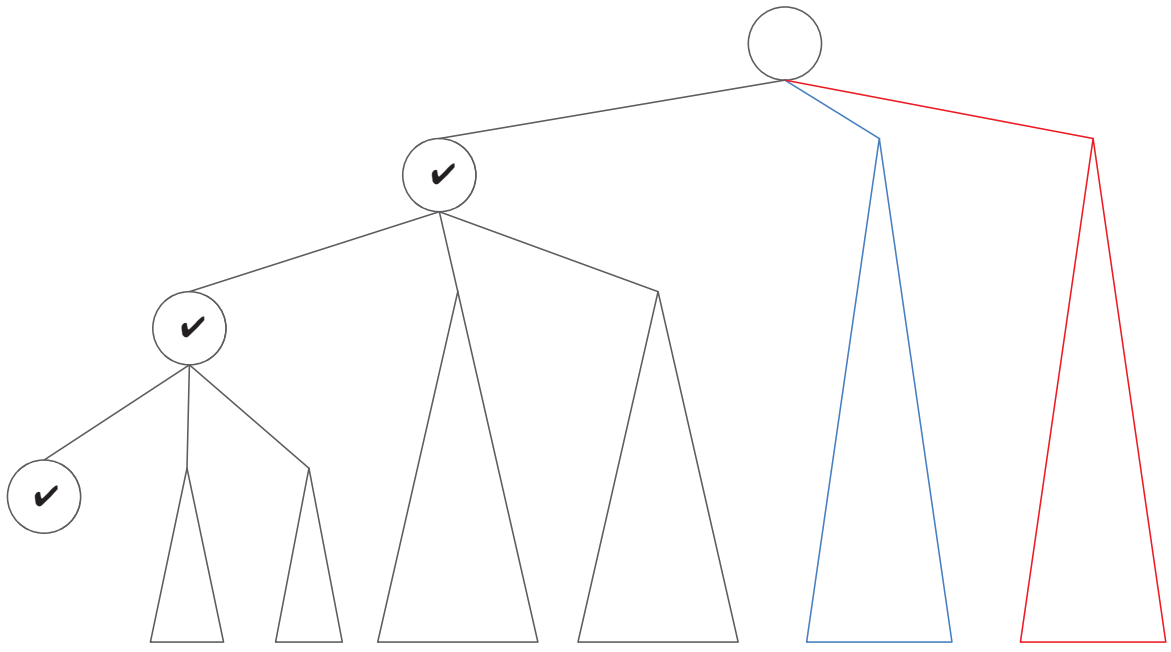
Principal Variation Splitting



Principal Variation Splitting



Principal Variation Splitting



Pseudocode (using openmp)

AlphaBeta → Principal Variation Splitting

```
AlphaBeta(state, α, β, depth)
    if depth == 0 then
        return evaluate(state)
    for each child of state
        value = -AlphaBeta(child, -β, -α, depth-1)
        if value >= β then
            return value
        α = max(α, value)
    return α
```

Command: g++ main.cpp -fopenmp

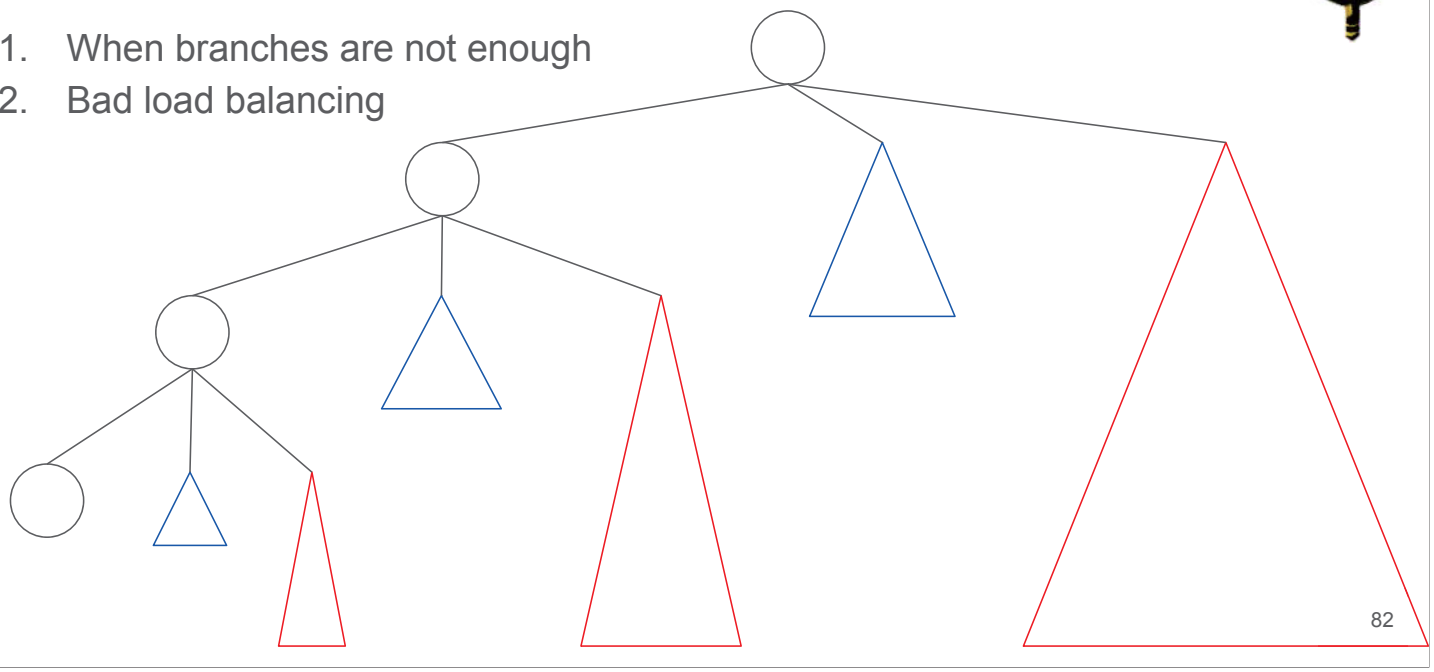
```
#include "omp.h"
PVS(state, α, β, depth)
    if depth == 0 then
        return evaluate(state)
    children = child_list(state)
    n = len(children)
    value = -PVS(children[0], -β, -α, depth-1)
    α = max(α, value)
    values = [None] * n

#pragma omp parallel for
    for (i = 1; i < n; i++) {
        values[i] = -AlphaBeta(children[i], -β, -α, depth-1)
    }

    for i = 1...n
        value = values[i]
        if value >= β then
            return value
        α = max(α, value)
    return α
```

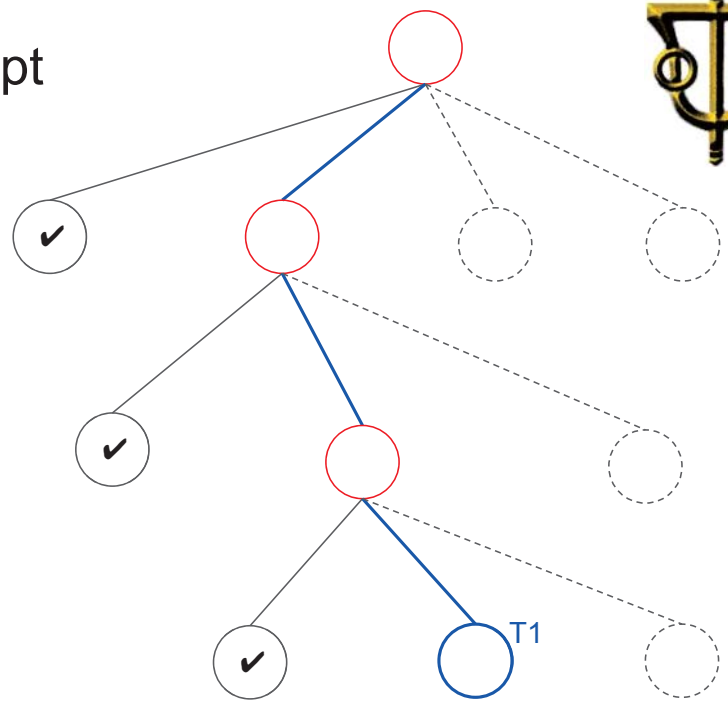

Principal Variation Splitting issue

1. When branches are not enough
2. Bad load balancing



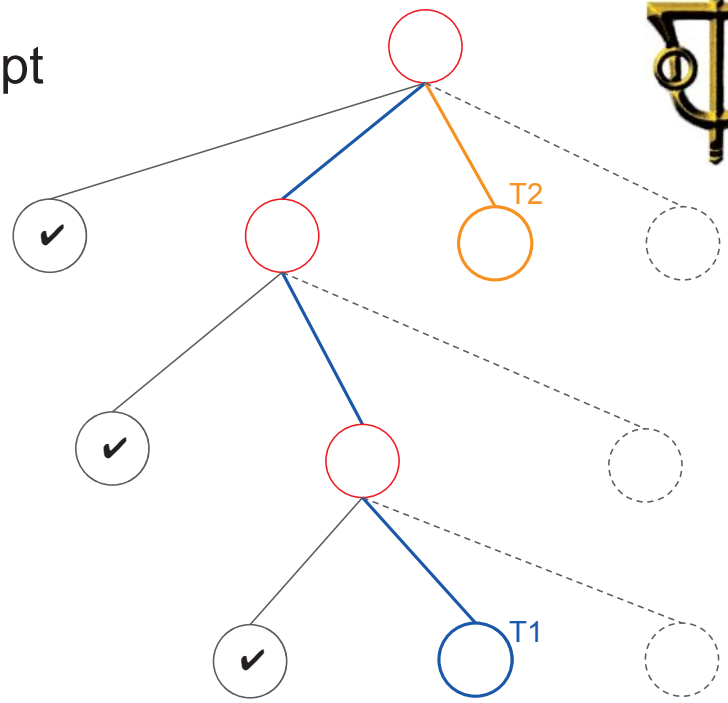
Young Brothers Wait Concept

- Split point
  - The search for first child has been completed
- Highest split point
  - Idle processor finds the highest split point to search
- Example
  - Thread 1
  - Thread 2 (idle)
  - Thread 3 (idle)
  - Thread 4 (idle)



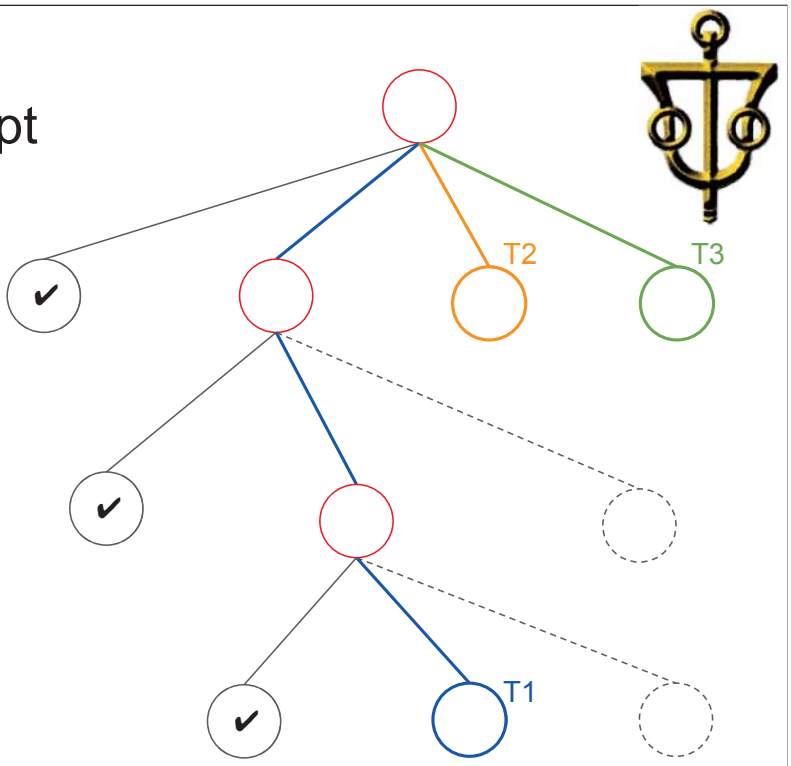
Young Brothers Wait Concept

- Split point
  - The search for first child has been completed
- Highest split point
  - Idle processor finds the highest split point to search
- Example
  - Thread 1
  - Thread 2
  - Thread 3 (idle)
  - Thread 4 (idle)



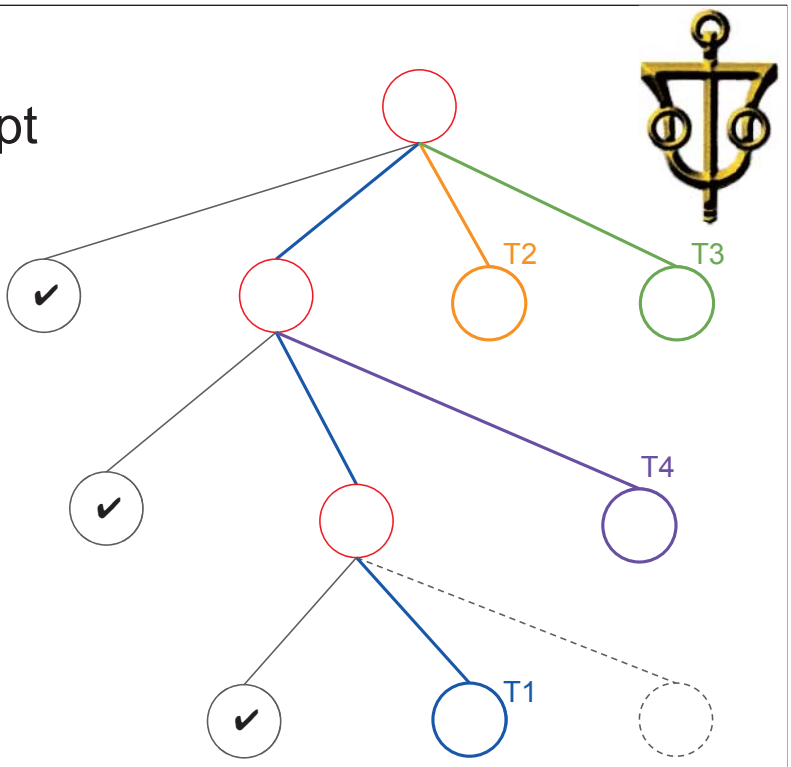
Young Brothers Wait Concept

- Split point
  - The search for first child has been completed
- Highest split point
  - Idle processor finds the highest split point to search
- Example
  - Thread 1
  - Thread 2
  - Thread 3
  - Thread 4 (idle)



Young Brothers Wait Concept

- Split point
  - The search for first child has been completed
- Highest split point
  - Idle processor finds the highest split point to search
- Example
  - Thread 1
  - Thread 2
  - Thread 3
  - Thread 4



Benchmark

Method	**Speed** (nodes/s)	**Ratio**
Single Thread	140,000,000	1.00
PVS (16 threads)	540,000,000	3.85
PVS (32 threads)	750,000,000	5.35
YBWC (16 threads)	1,250,000,000	8.92
YBWC (32 threads)	2,230,000,000	15.92

Outline

1. Introduction
2. Action generator
3. Evaluation function
4. Search algorithm
5. Parallel search algorithm
6. Opening book
7. Conclusion

Opening book

Store the search result in advance

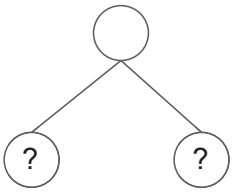
- Required
  - A Search Engine which can evaluate given states
    - GLEM + alphabeta
    - policy-value network + MCTS (i.e. AlphaZero)
- Optional
  - Expert games
  - Other strong engines

Opening book

Method

1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save

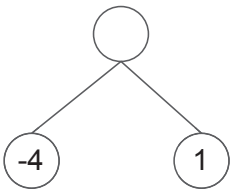
Opening book



Method

1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save

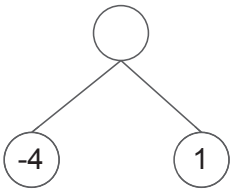
Opening book



Method

1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save

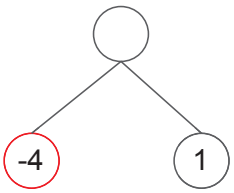
Opening book



Method

1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save

Opening book



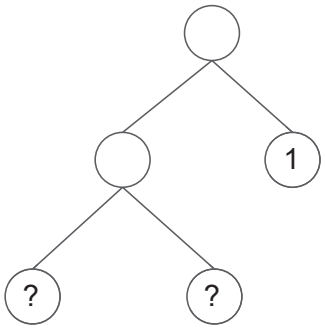
Method

1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save

Opening book

Method

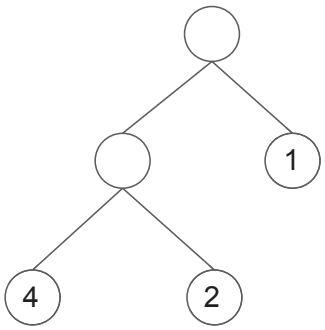
1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save



Opening book

Method

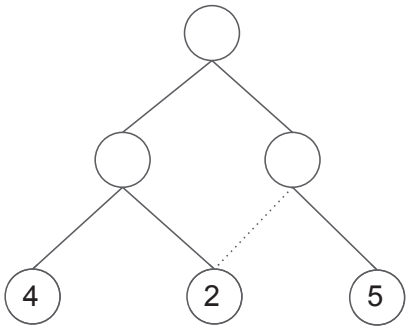
1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save



Opening book

Method

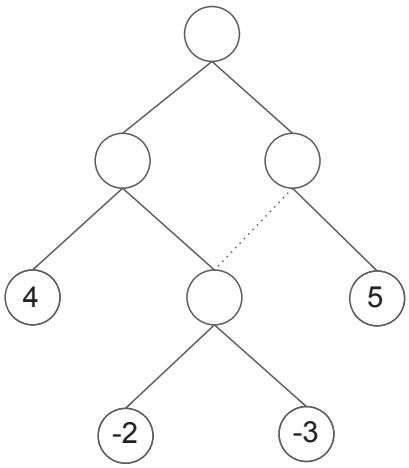
1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save



Opening book

Method

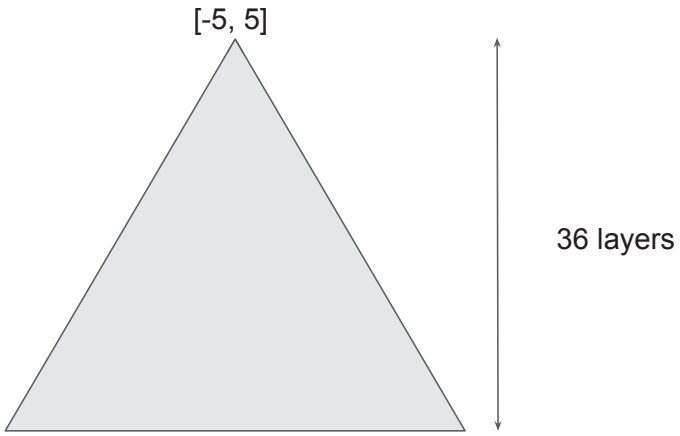
1. Select
   - Minimax, MCTS or Game Trajectory
2. Expand all children
3. Search for all children
4. Save



…

My big opening book

- Width: [-5, 5]
- Depth: 36
  - My program, Maverick, can search the remaining 24 layers of the game tree in 1 second.



Outline

1. Introduction
2. Action generator
3. Evaluation function
4. Search algorithm
5. Parallel search algorithm
6. Opening book
7. Conclusion

幫大家整理一下

- 走步生成
  - Kogge-Stone + AVX2：約140000000 nodes/s
- 估值函數
  - GLEM：單層網路，兼具效能的精確估 值
- 算法優化
  - Null window
  - Narrow window
  - MPC：基於數學統計結果，97.5%的信心來減少搜尋深度
- 多執行緒
  - 更好的附載平衡：Root Parallel ➔ PVS ➔ YBWC
- 開局庫
  - 大局觀不輸AlphaZero

Thanks for your attention