Chess-AI Parallel Programing

組員: 蔡承恩 310551068
蔡旻樺 310553049
孔憲文(找不到人)

Layout

Introduction

Chess Program Layout

Problem

Costs Exponential growth as depth increase

Method

Alpha-Beta Pruning

Nega search

PVS search

Correctness

Game with Stockfish

Evaluation

With(out) Alpha-Beta Pruning

Threads and Performance

Nega search vs PVS search

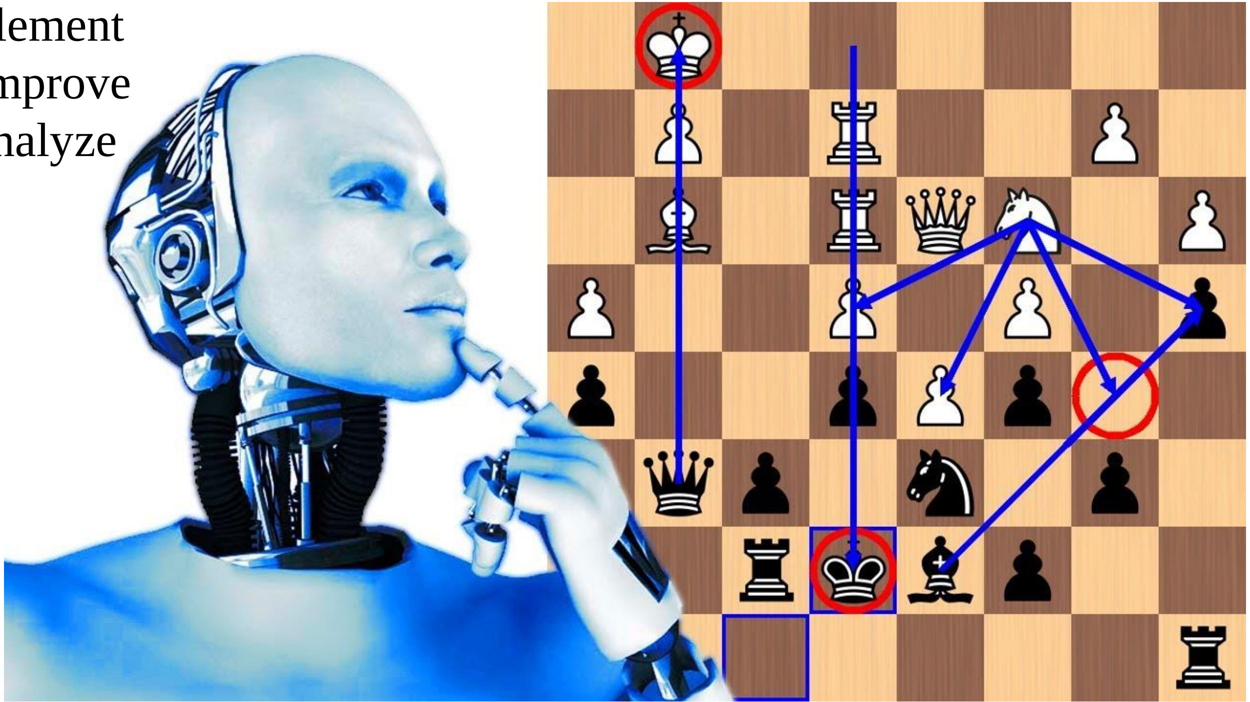
Conclusion

Introduction

In a chess match in 1997, the parallel chess supercomputer Deep Blue defeated the world champion Garry Kasparov making a milestone in the history of computer science.

Introduction

We would like to study the secret behind Deep Blue and implement different parallel ways to improve the Chess AI engine, and analyze the performance by time, scalability and portability.

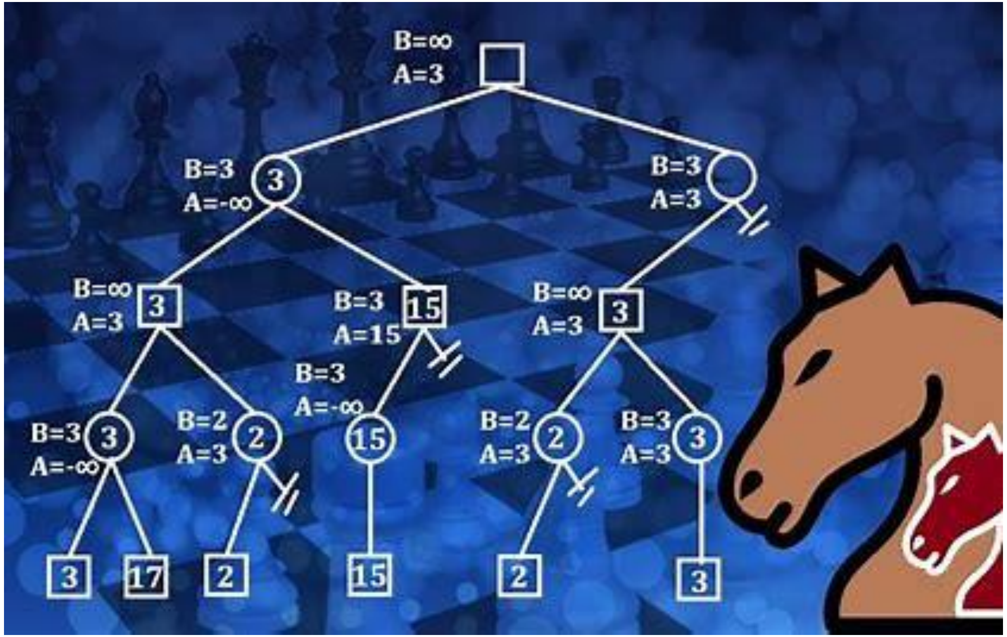


Introduction

We use **Alpha-Beta Pruning** to eliminate the need to search for meaningless branches.

And our biggest goal is to use Alpha-Beta Pruning as our search method, and use OpenMP to parallelize and improve performance.

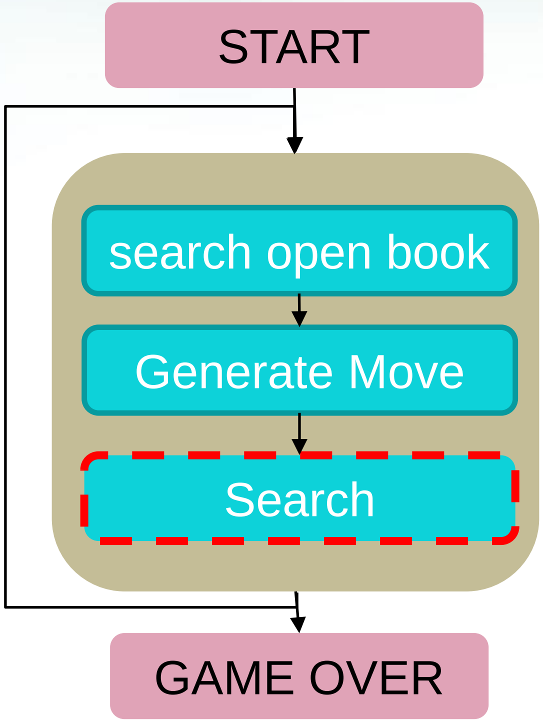
The algorithms that implement parallelization are, only use Alpha-Beta Pruning's algorithm and the algorithm designed for parallelization, PVS Search.



Problem

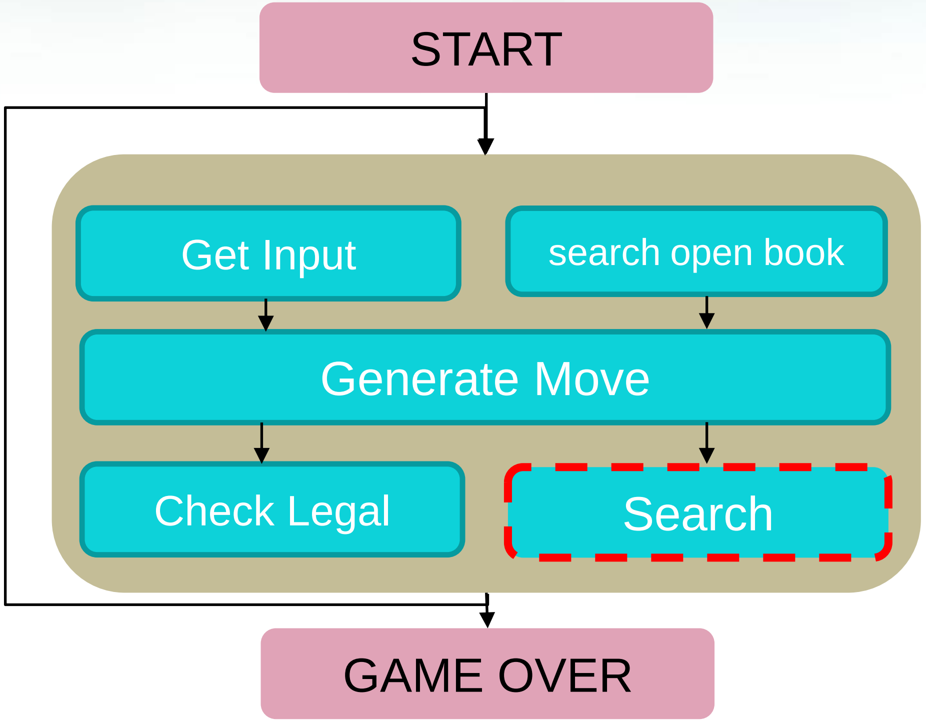
- The difference between whether to use Alpha-Beta pruning .
- Does Alpha-Beta pruning seriously lose scalability?
- Only use Alpha-Beta Pruning's algorithm (Method I)
  or PVS Search, which is better?
- Are there other ways to make performance better

Method



(If both are computers)

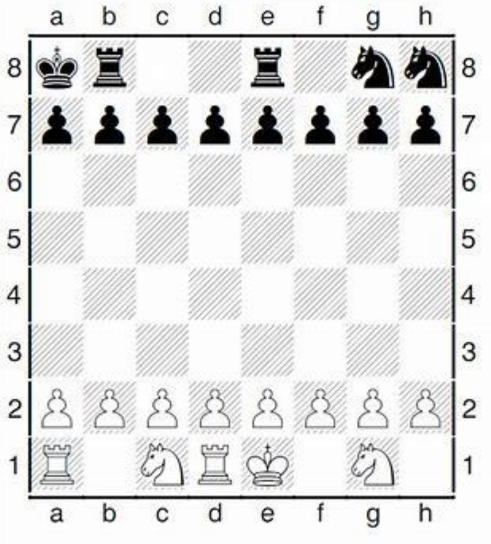
hply++
side ^= 1



(If both are player & computer)

Method – Search Open Book

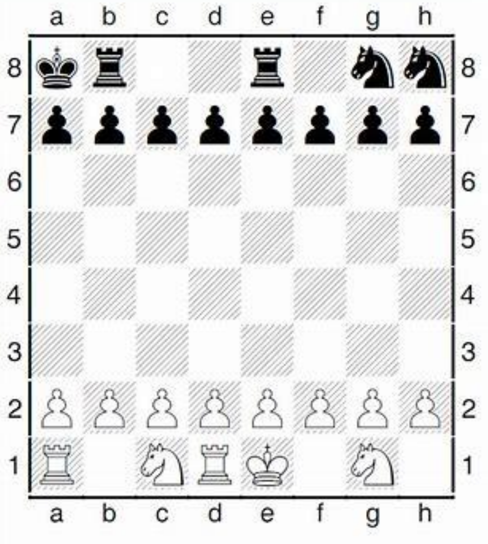
1. **g1f3 g8f6 c2c4 b7b6 g2g3**
2. **g1f3 g8f6 c2c4 c7c5 b1c3 b8c6**
3. **g1f3 c7c5 c2c4 b8c6**
4. **c2c4 g8f6 b1c3 c7c5**
5. **d2d4 g8f6 g1f3 c7c5**



Assuming this is the content of the whole book.

Method – Search Open Book

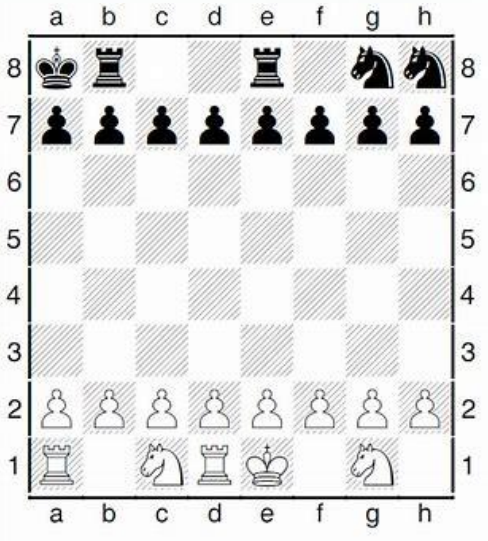
1. **g1f3 g8f6 c2c4 b7b6 g2g3**
2. **g1f3 g8f6 c2c4 c7c5 b1c3 b8c6**
3. **g1f3 c7c5 c2c4 b8c6**
4. ~~**c2c4 g8f6 b1c3 c7c5**~~
5. ~~**d2d4 g8f6 g1f3 c7c5**~~



We choose randomly, suppose we choose **g1f3** in the first step from the whole book

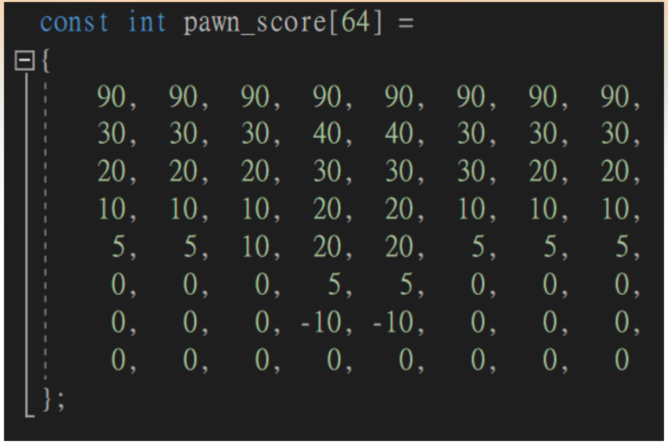
Method – Search Open Book

1. ~~g1f3 g8f6 c2c4 b7b6 g2g3~~
2. ~~g1f3 g8f6 c2c4 c7c5 b1c3 b8c6~~
3. g1f3 c7c5 c2c4 b8c6
4. ~~c2c4 g8f6 b1c3 c7c5~~
5. ~~d2d4 g8f6 g1f3 c7c5~~



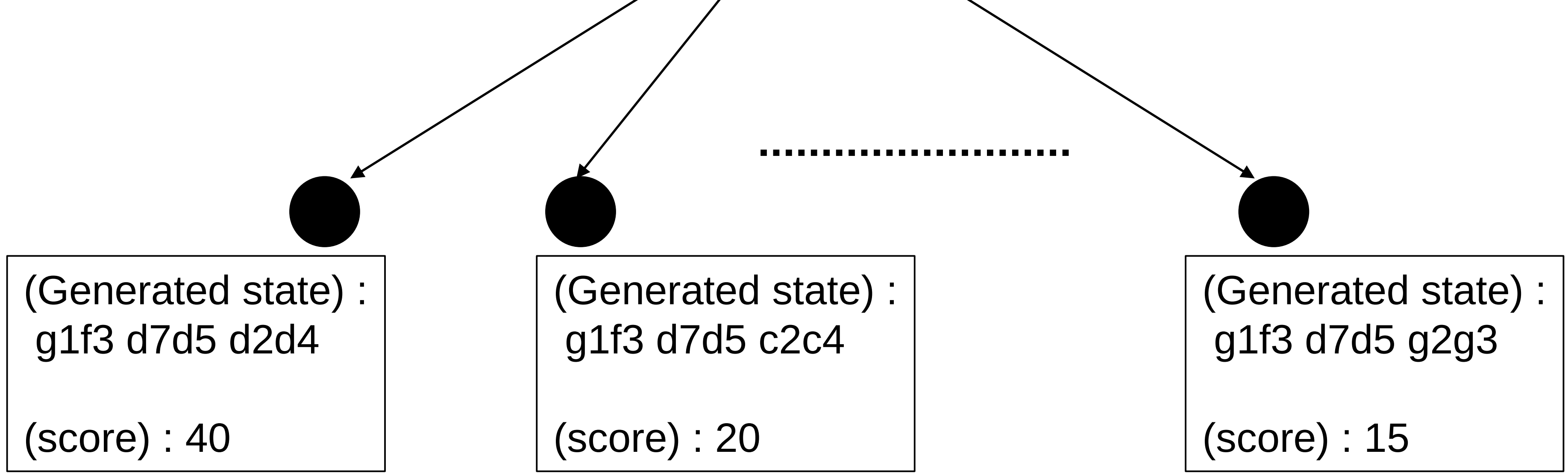
We choose randomly, suppose we choose **c7c5** in the second step in the range 1 to 3

Method – Generate Move



```
const int pawn_score[64] =
{
     90,  90,  90,  90,  90,  90,  90,  90,
     30,  30,  30,  40,  40,  30,  30,  30,
     20,  20,  20,  30,  30,  30,  20,  20,
     10,  10,  10,  20,  20,  10,  10,  10,
      5,   5,  10,  20,  20,   5,   5,   5,
      0,   0,   0,   5,   5,   0,   0,   0,
      0,   0,   0, -10, -10,   0,   0,   0,
      0,   0,   0,   0,   0,   0,   0,   0
};
```

(current state) : g1f3 d7d5
(current score) : 10



(Generated state) :
g1f3 d7d5 d2d4

(score) : 40

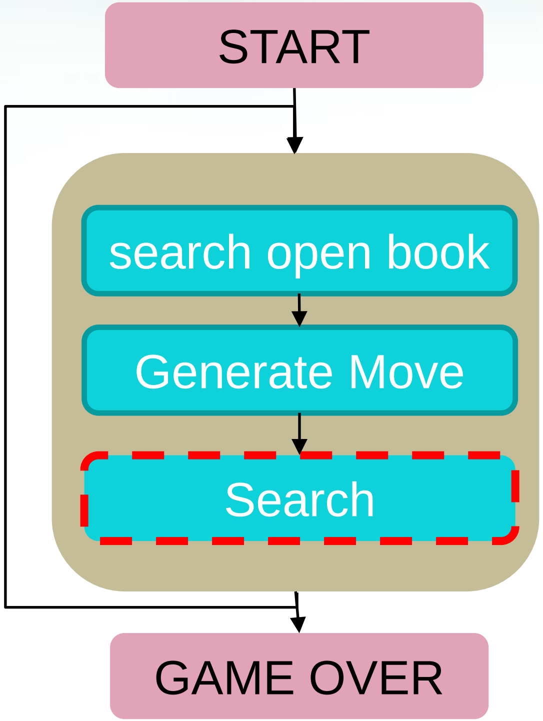
(Generated state) :
g1f3 d7d5 c2c4

(score) : 20

(Generated state) :
g1f3 d7d5 g2g3

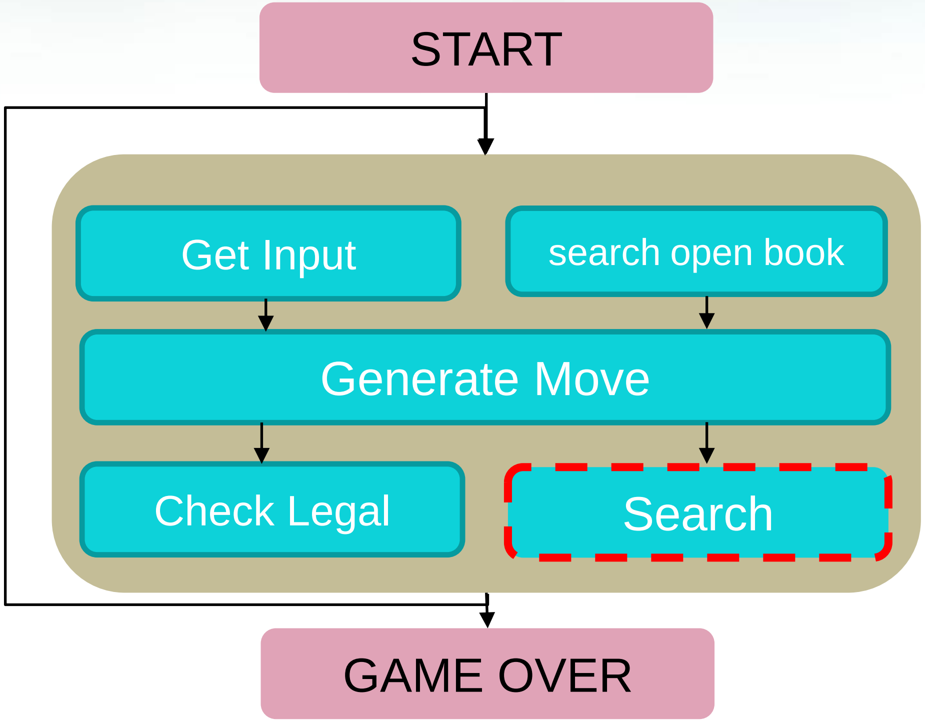
(score) : 15

Method



(If both are computers)

hply++
side ^= 1

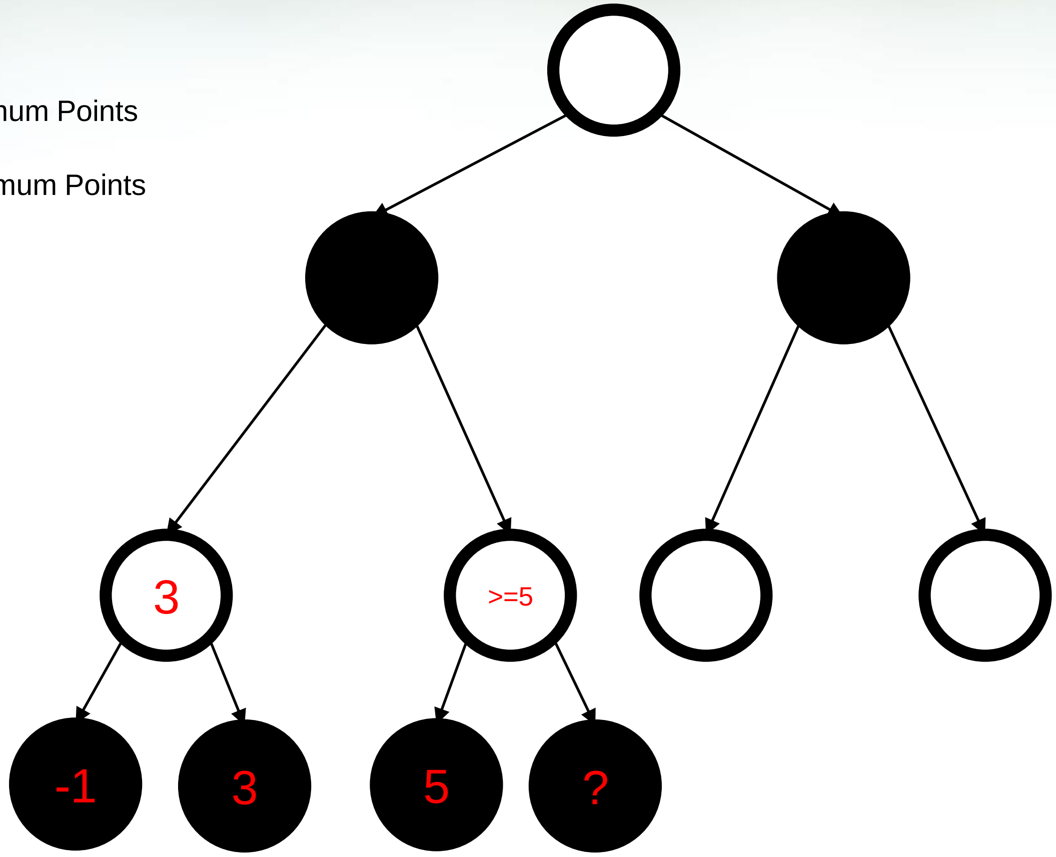


(If both are player & computer)

Method - Cutoff

● Search minimum Points

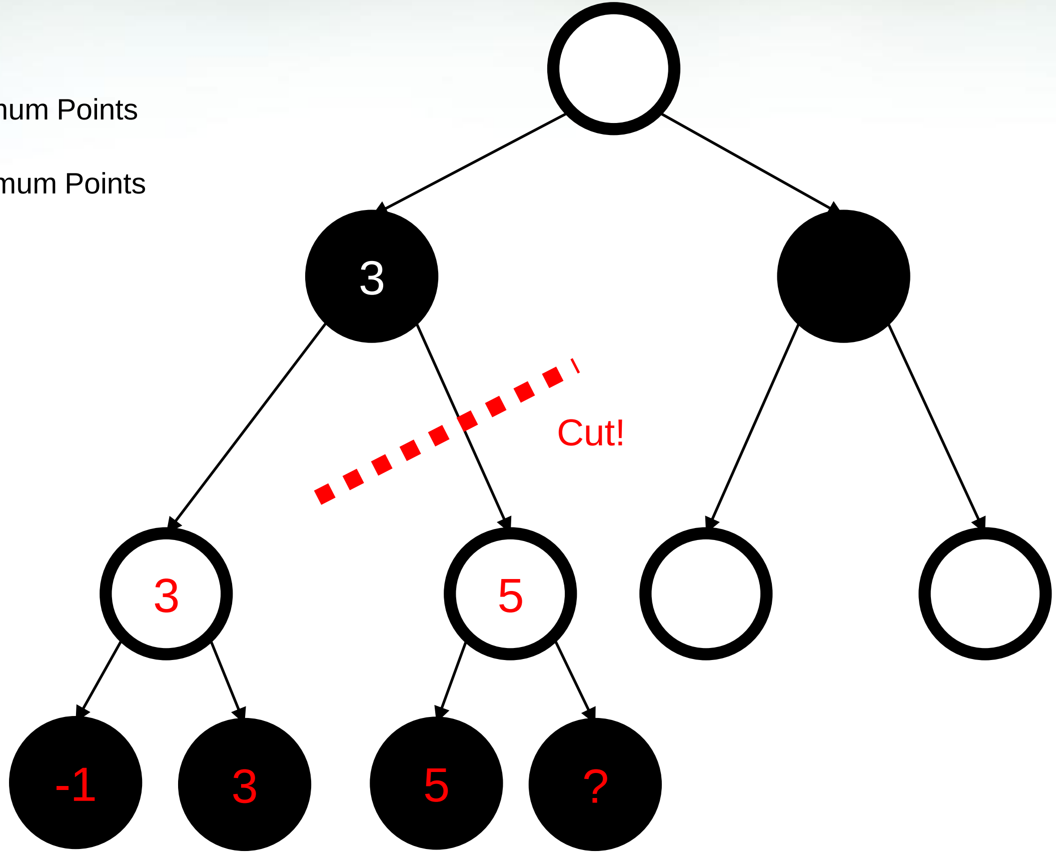
○ Search maximum Points



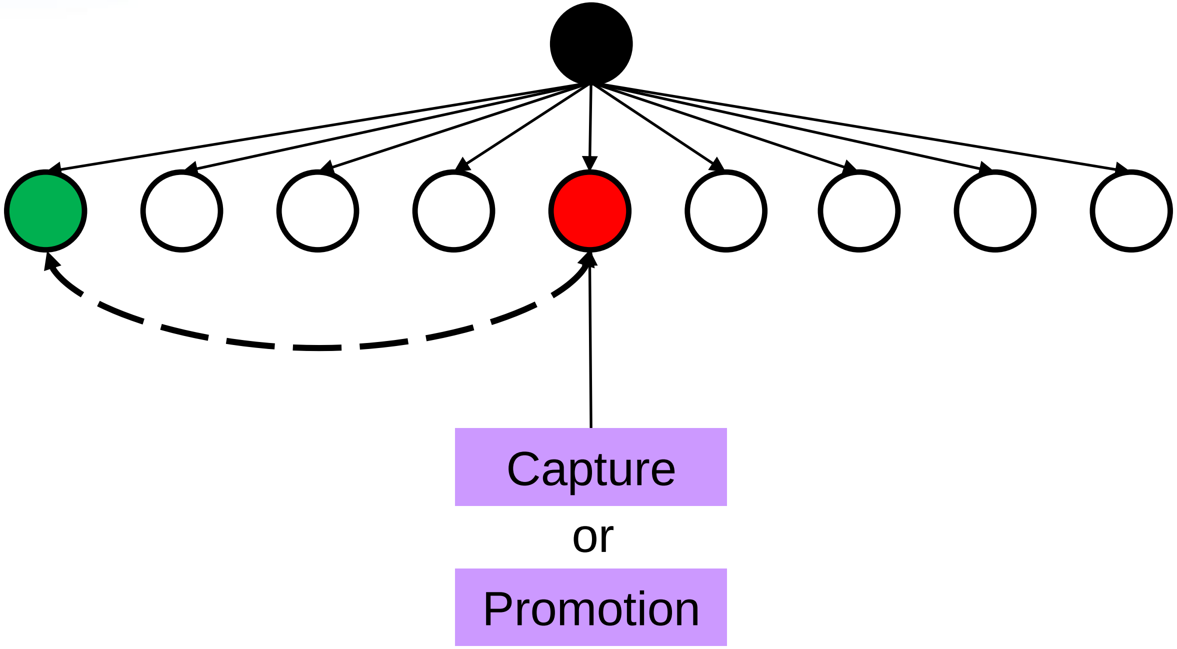
Method - Cutoff

● Search minimum Points

○ Search maximum Points



Method - Sort

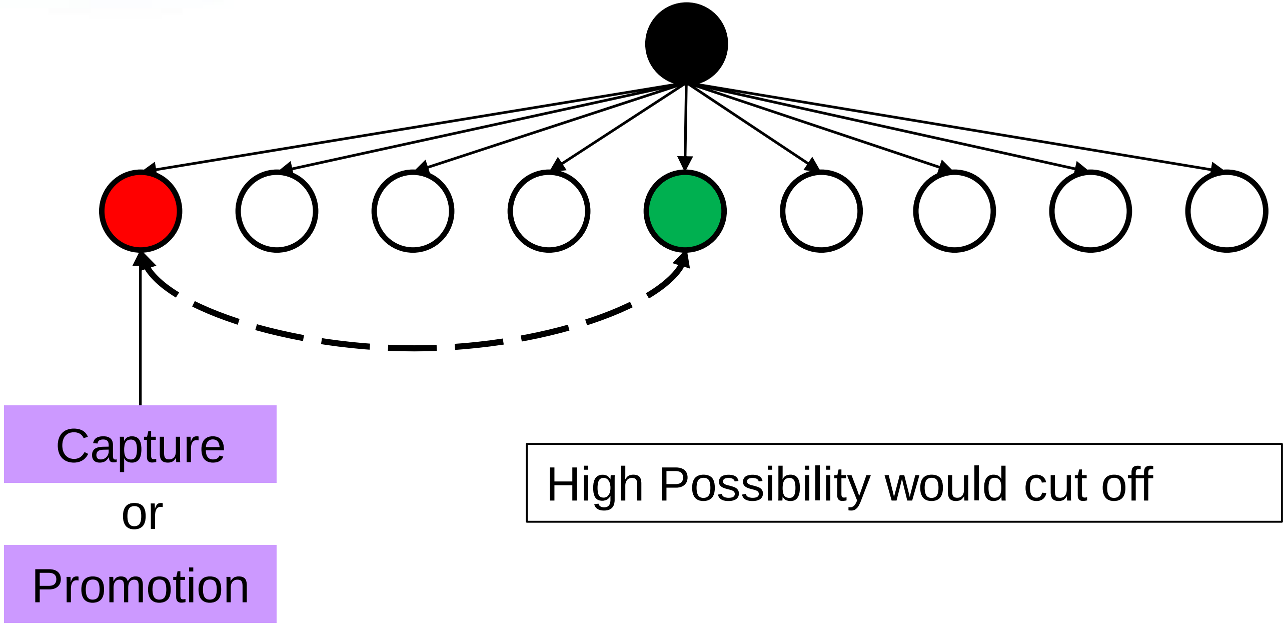


Capture

or

Promotion

Method - Sort



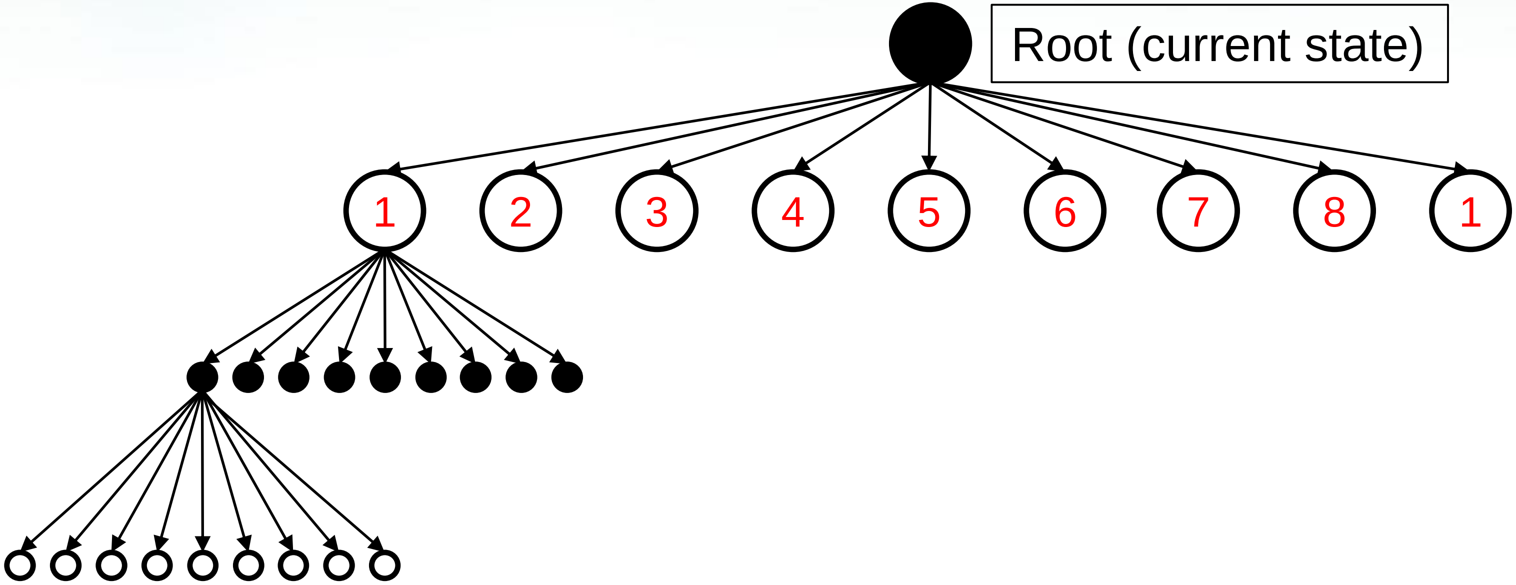
Capture

or

Promotion

High Possibility would cut off

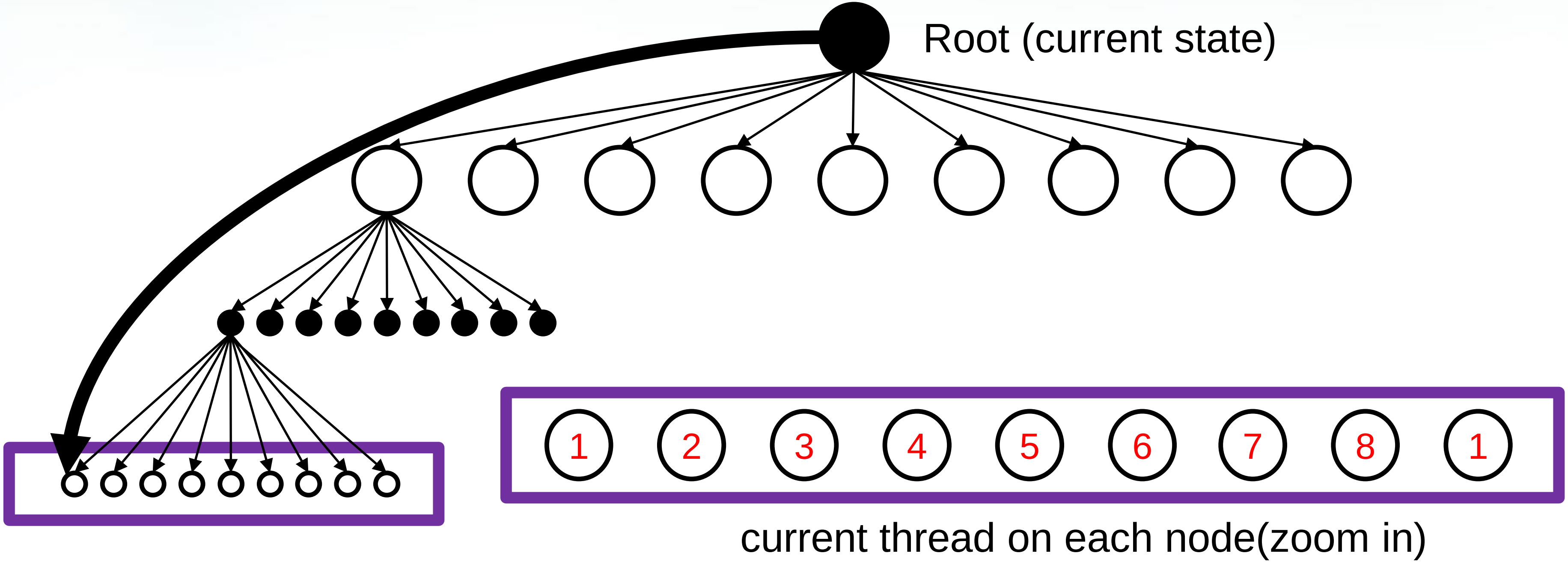
Method I



Root (current state)

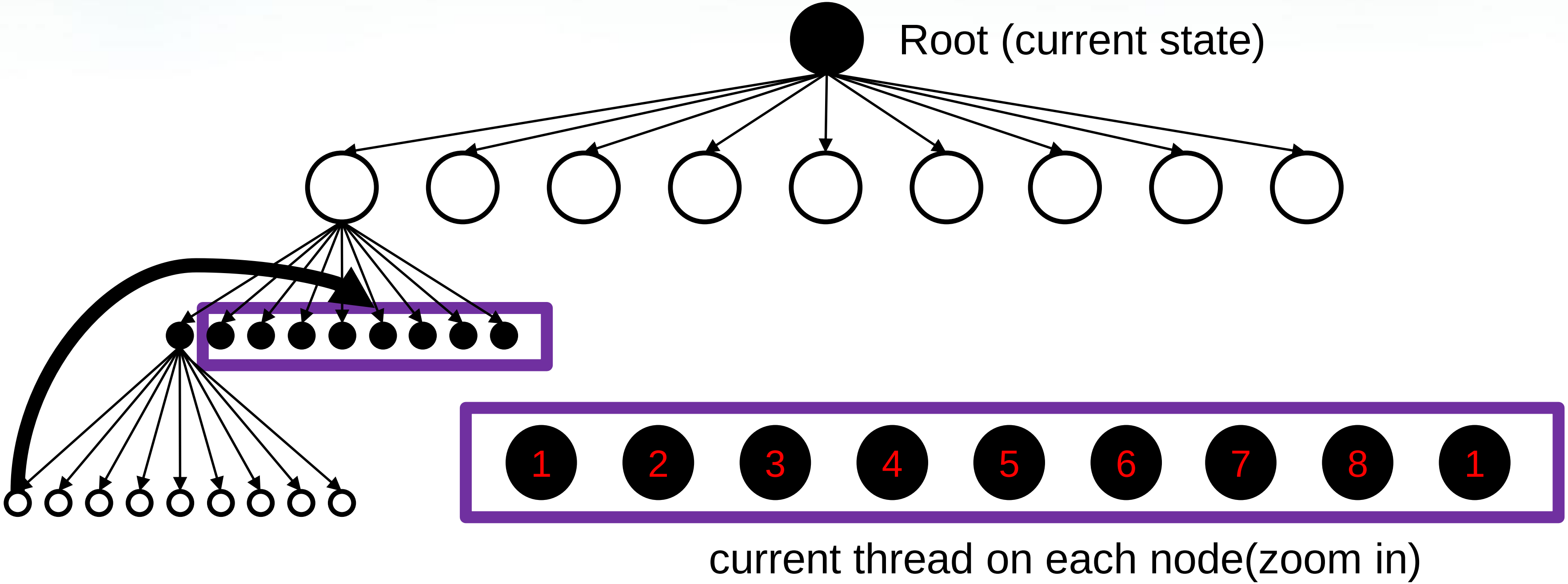
Each node present one board state

Method II– PVS Search (alpha-beta)



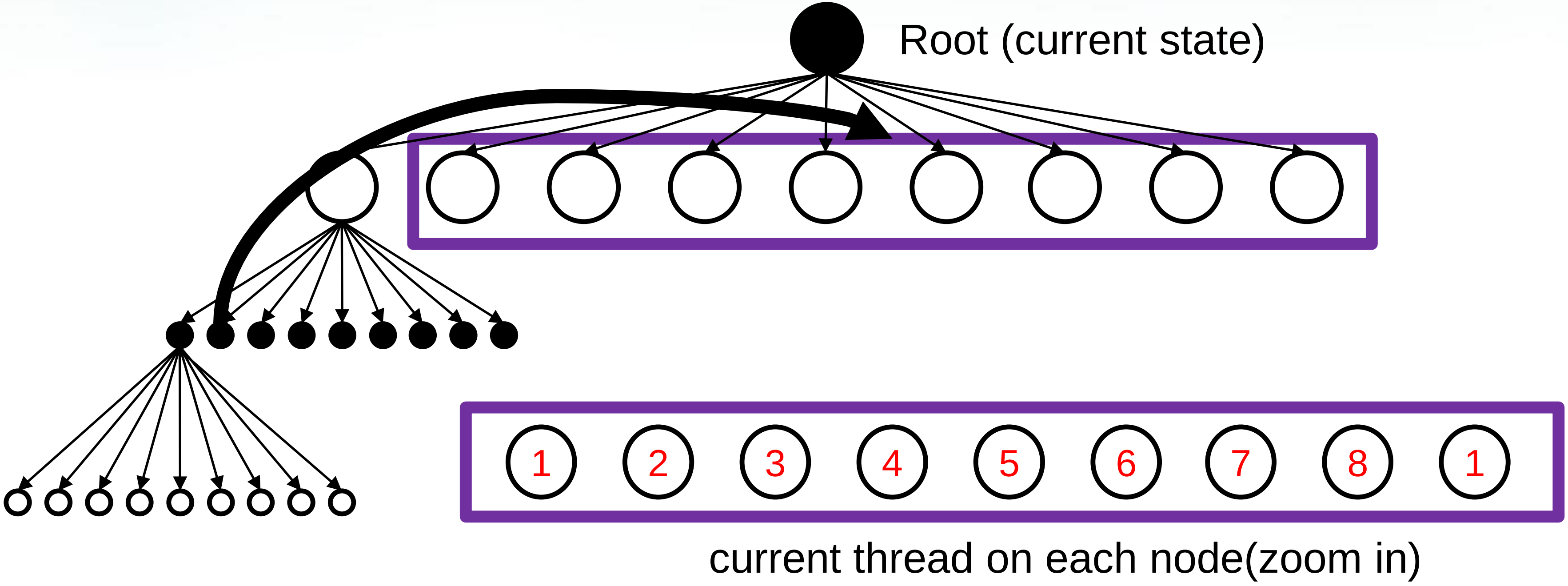
Each node present one board state

Method – PVS Search (alpha-beta)



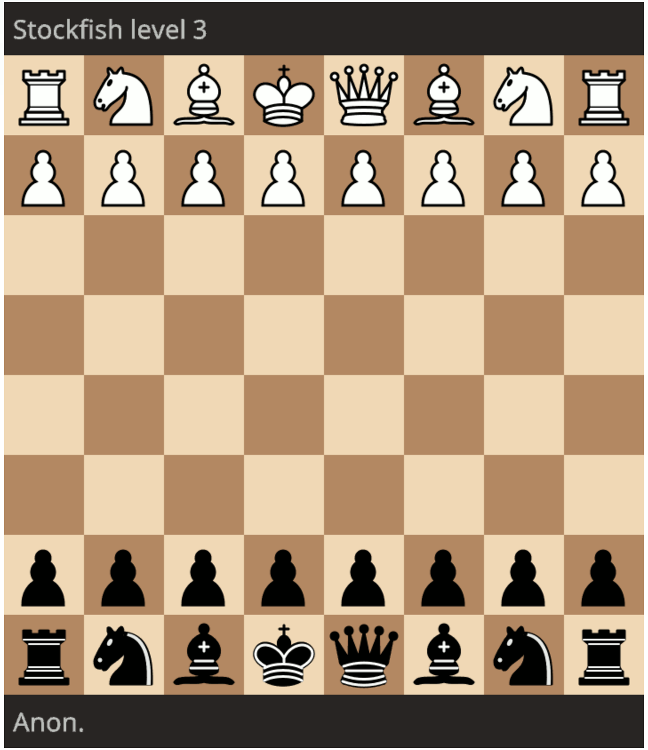
Each node present one board state

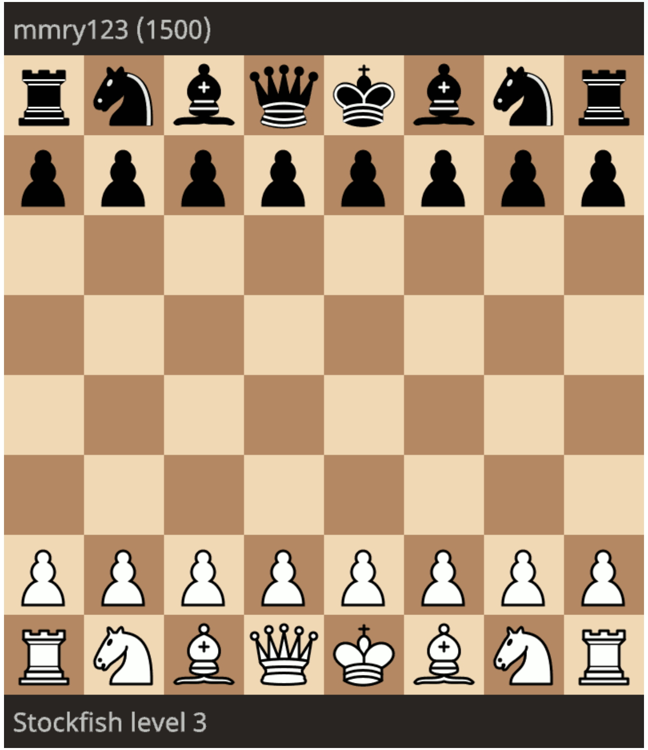
Method – PVS Search (alpha-beta)



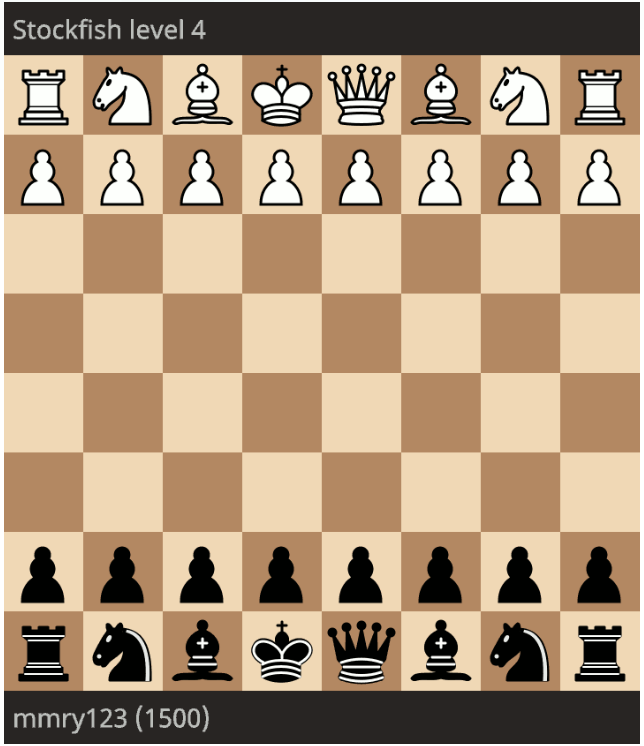
Each node present one board state

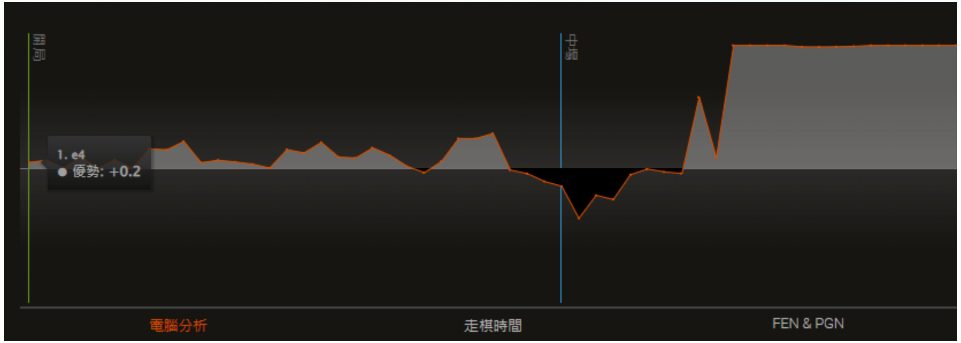
Correctness

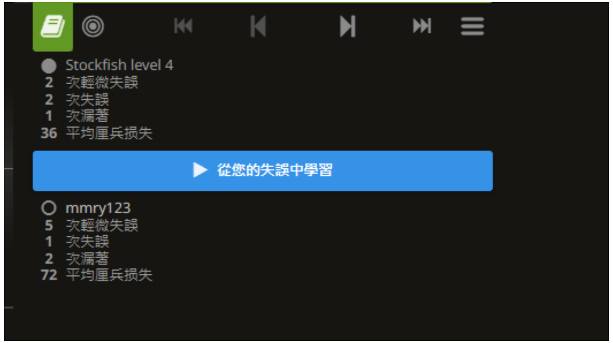




Correctness







● Stockfish level 4
2 次輕微失誤
2 次失誤
1 次漏著
36 平均厘兵損失

▶ 從您的失誤中學習

○ mmry123
5 次輕微失誤
1 次失誤
2 次漏著
72 平均厘兵損失

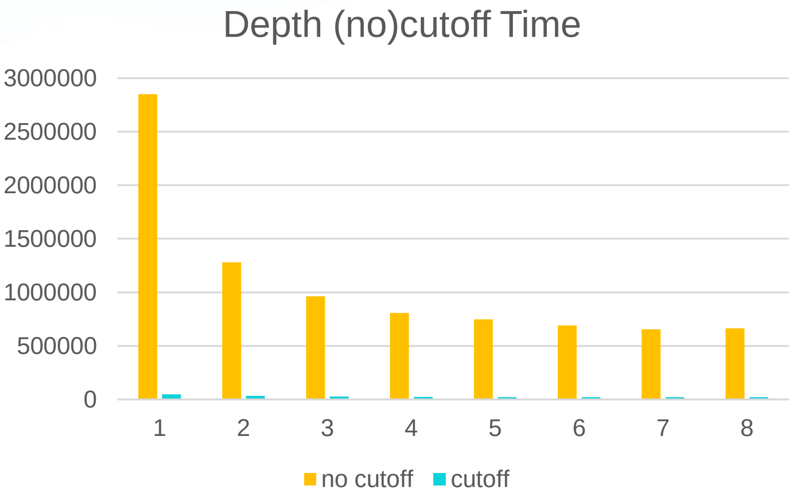
Platform & Device

Window 10, 64bits

Intel® Core™ I7-6700 CPU @ 3.40GHz (4CPUs 8Threads)

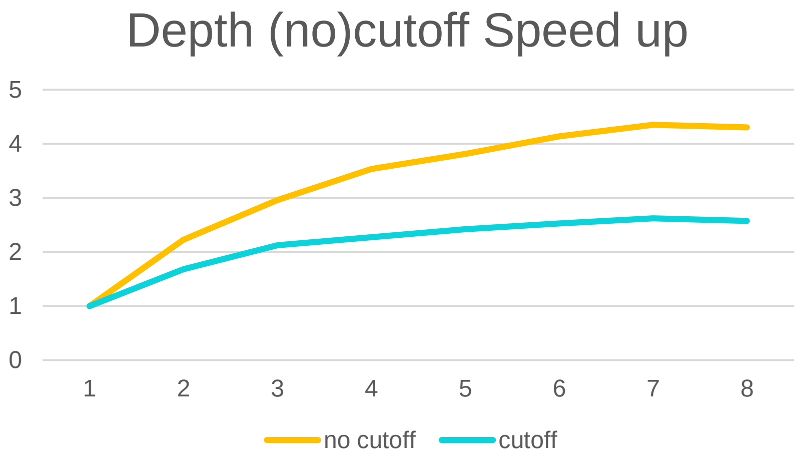
RAM 16GB

Evaluation



Depth (no)cutoff Time

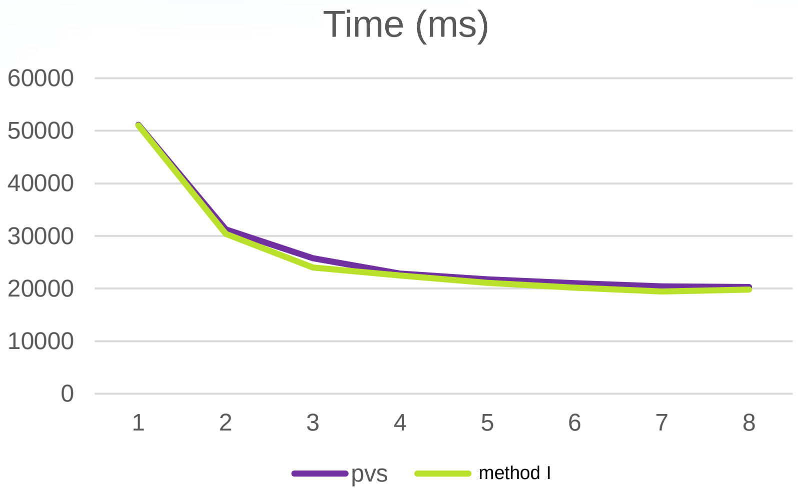
Time

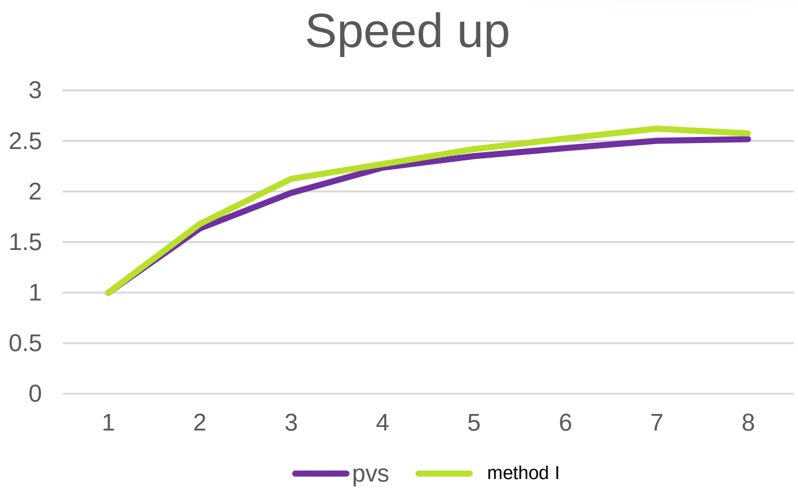


Depth (no)cutoff Speed up

Scalability

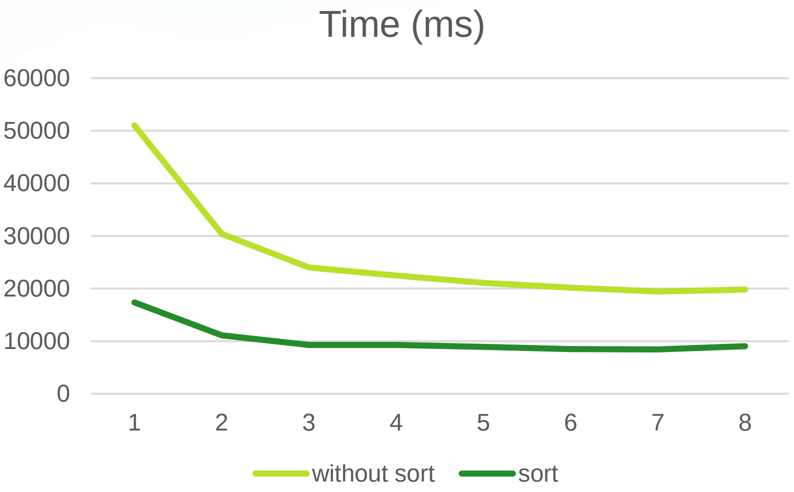
Evaluation Depth = 6, hply = 5

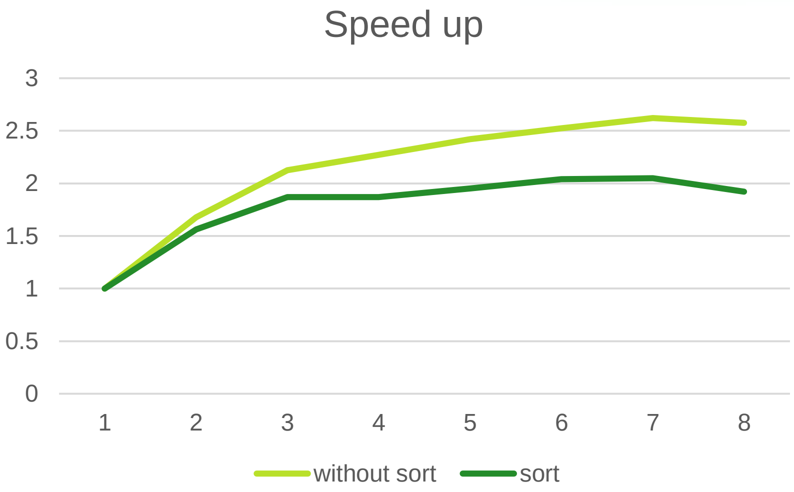




without sort

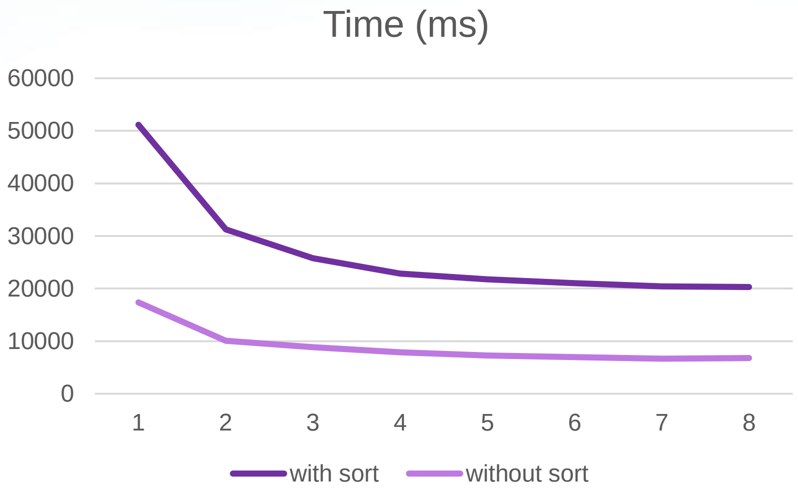
Evaluation Depth = 6, hply = 5

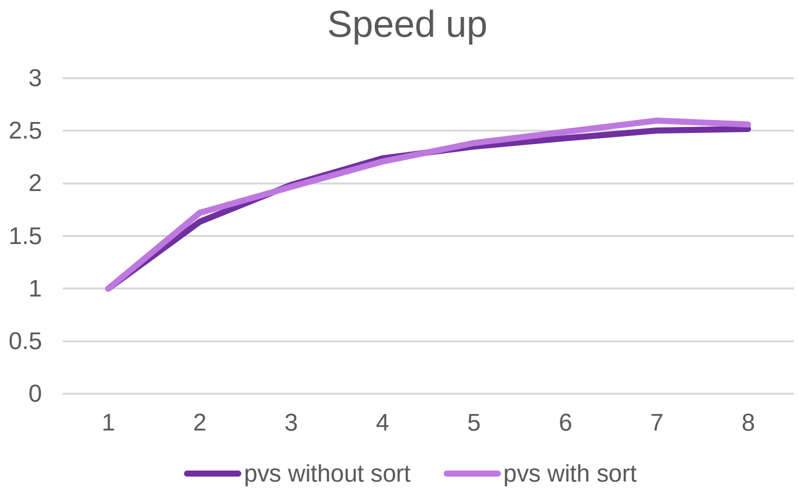




Method I

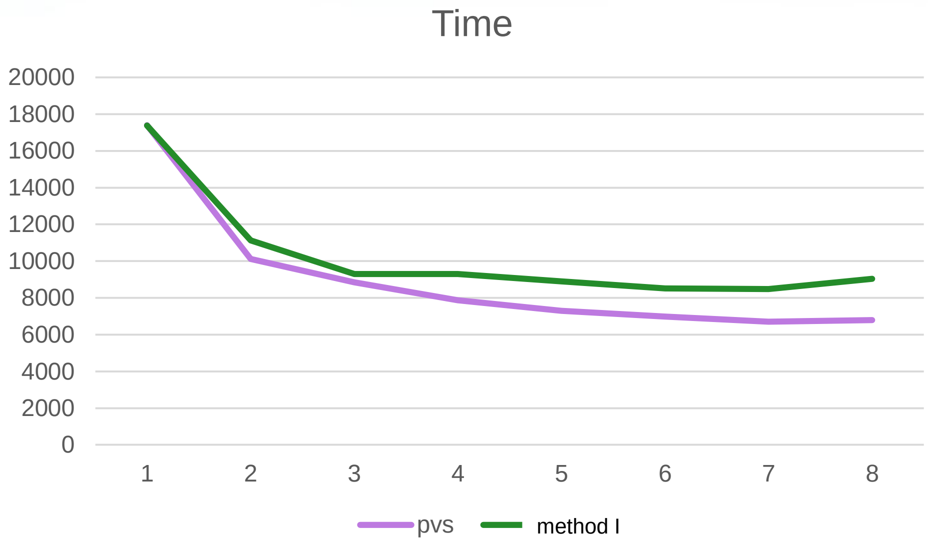
Evaluation Depth = 6, hply = 5





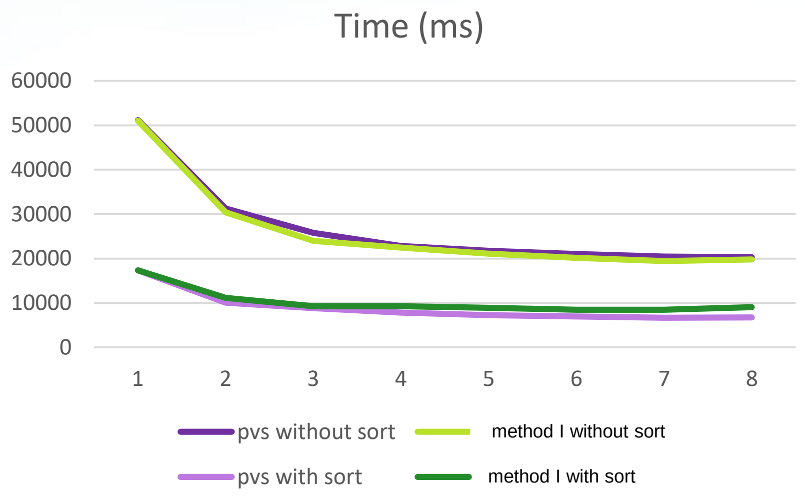
PVS search

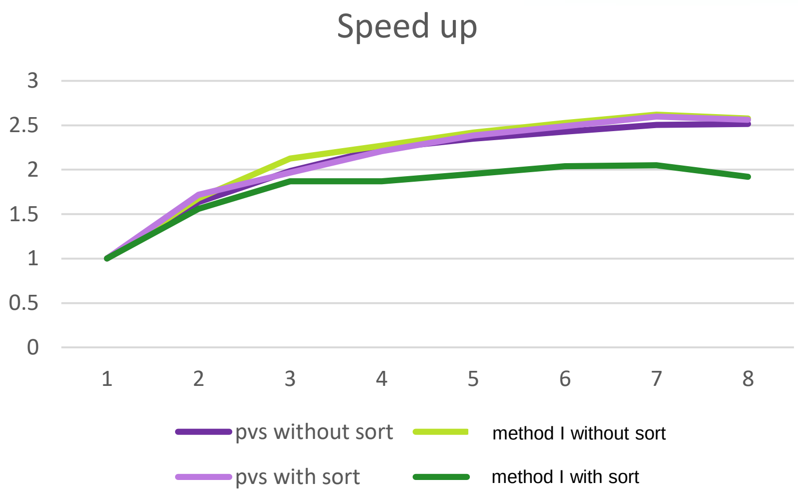
Evaluation Depth = 6, hply = 5



With sort

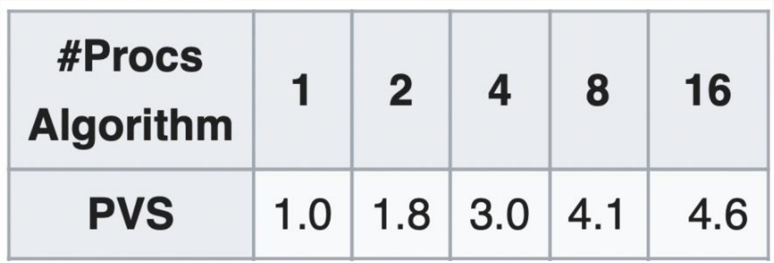
Evaluation Depth = 6, hply = 5





Evaluation

Thread	performance
1	1
2	0.86
3	0.66
4	0.55
5	0.48
6	0.42
7	0.37
8	0.32



#Procs Algorithm	1	2	4	8	16
PVS	1.0	1.8	3.0	4.1	4.6

Thread	performance
1	1
2	0.9
4	0.75

Performance :
(Speed up) / threads number

Conclusion

- Q : **The difference between whether to use Alpha-Beta pruning .**
  - The time required to use Alpha-Beta Pruning varies greatly.

- Q : **Only use Alpha-Beta Pruning's algorithm (Method I) or PVS Search, which is better?**
  - When we compare PVS Search and Method I, we find that there is no significant difference in the time spent and performance improvement between the two. We guess that because PVS Search allocates threads, it will need to wait until all threads have completed the calculation of the current layer. The next distribution will be carried out, which will offset its advantage of evenly distributed cut-off.

Conclusion

- Q : **Does Alpha-Beta pruning seriously lose scalability?**
  - We found that SpeedUp also increased significantly as the thread increased from 1 to 4, but when the thread was increased from 5, the increase was not so obvious.
  - We only have 4 cores, but one core can generate two threads. When the second thread is performing certain calculations, it can only be used when the first thread is idle, so the acceleration effect is limited.
  - We feel that if we look at threads1~4, the scalability when using Alpha-Beta Pruning is as we expected.

Conclusion

- Q：**Performance improvement after sorting**?
  - After nodes in Method I was sorted, the search speed was much faster than without the sort, but the speed up did not increase according to the number of threads (performance decreased).
  - After nodes in PVS Search was sorted, the time spent is significantly reduced due to the earlier truncation, and the performance is still the same as the untruncated due to the increase in threads.

Thanks for Listening~