Rapport TD2 - Optimisation

Par Stanislaw PLUSZKIEWICZ et Ellmo RIVETTI

Objectif du TD:

L'objectif du TD est de modifier le code de TSCP afin d'optimiser les structures de données utilisées.
L'optimisation des structures va résulter en une amélioration des performances du programme car nous allons remanier les méthodes parcourant ces structures afin d'effectuer moins d'itérations.

Réalisation du TD:

Déroulement:

Nous avons commencé le développement du TD lors des séances prévues. Durant ces séances, nous avons commencé à travailler sur l'optimisation 1.
Lors de notre travail personnel, nous avons décidé de laisser de côté l'optimisation 1 le temps d'effectuer l'optimisation 2 car celle-ci nous paraissait plus simple.

Il nous a fallu environ 1h30 afin de réaliser l'optimisation 2 plus 15 minutes à la fin lors du merge des optimisations afin de résoudre un bug que nous n'avions pas vu.

Nous avons mis beaucoup plus de temps pour réaliser l'optimisation 1. Soit:

- 45 minutes lors du premier TD
- 2h lors du second TD (ou nous avons principalement fait la méthode sync_board())
- 6h en travail personnel chacun

Chaque optimisation a été développée sur une branche spécifique de notre GitHub. Une fois que les deux furent finies, nous avons merge ces branches sur *main* (la branche principale du projet Git).

Branches:

- Optimisation 1
- Optimisation 2
- Merge des deux optimisations

Remarque:

L'état de l'optimisation 2 sur la branche opti2 n'est pas final. En effet, nous avons rencontré un bug lors du merge avec l'optimisation 1. Celui-ci est donc résolu sur la branche main.

Problèmes rencontrés

Lors du développement du TD, nous avons rencontré plusieurs problèmes.
La première difficulté, à été de bien saisir le fonctionnement de TSCP. Plus particulièrement, les structures de données qui ont été utilisées.

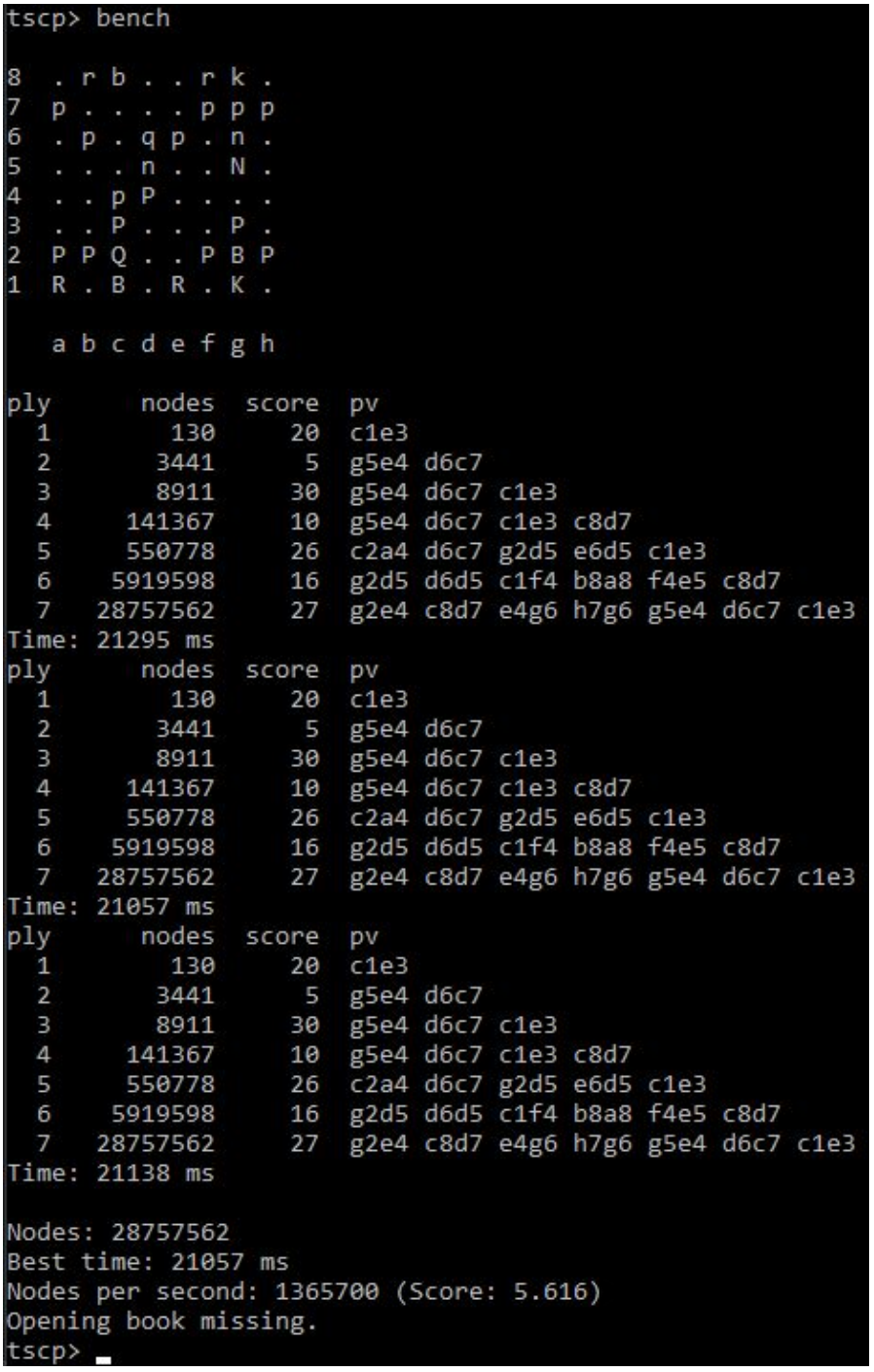
Le second problème rencontré a été l'écriture des méthodes *takeback()* et *makemove()* de l'optimisation 1.
Les changements apportés dans ces méthodes furent complexes à ajouter. En effet, il y avait plusieurs cas spéciaux comme la prise en passant ou bien la promotion qu'il a fallu gérer. Ces cas spéciaux ont alors causé plusieurs erreurs lors des tests.

Le debug fut aussi un aspect complexe du développement. En effet, le langage C ne permet pas l'implémentation de tests poussés. Grâce au debugger de Visual Studio 2019, nous avons pu alors vérifier les valeurs de nos variables et ainsi analyser les changements lors du déroulement du programme.
Nous avons aussi écrit des méthodes de vérification comme *checkBoard()* qui nous a permis de vérifier l'état du plateau à chaque étape.
Nous appelions cette méthode au début et à la fin de chaque méthode afin de vérifier si celle-ci était la cause d'une modification non désirée.

Résultat - Benchmarks

Résultat initial de TSCP

Afin de tester notre code, nous avons utilisé la méthode *bench()* de TSCP. Lors de ces benchs, nous avons pris une profondeur de 7.
Tous les tests ont été effectués sur le même PC afin qu'il n'y ai pas de changement dû aux différentes puissances de calcul des appareils. De plus, chaque bench est effectué en mode *Release* et en *x64*.
Voici les résultats obtenus via le code original de TSCP:



Optimisation 1

Cette optimisation consiste à ajouter des structures de données permettant d'accéder de manière directe aux positions des pièces des deux joueurs. Ceci évite de vérifier les 64 cases de l'échiquier à chaque vérification.

L'optimisation 1 nous a permis de gagner **5.292 secondes** sur les valeurs originales de TSCP. De 21057 ms à 15765 ms.

TSCP parcours 1365700 nodes par secondes et l'optimisation 1, 2094784 nodes par secondes.

Nous obtenons donc un score de **8.615** soit environ **+2.999** par rapport à TSCP.

```
tscp> bench

8  . r b . . r k .
7  p . . . . p p p
6  . p . q p . n .
5  . . . n . . N .
4  . . p P . . . .
3  . . P . . . P .
2  P P Q . . P B P
1  R . B . R . K .

   a b c d e f g h

ply      nodes  score  pv
  1        131     20  c1e3
  2       4717      5  g5e4 d6c7
  3      10487     30  g5e4 d6c7 c1e3
  4     184442     10  g5e4 d6c7 c1e3 c8b7
  5     647714     26  c2a4 d6c7 g2d5 e6d5 c1e3
  6    8400851     16  g2d5 d6d5 c1f4 b8a8 f4e5 c8b7
  7   33024284     27  g2e4 c8d7 e4g6 h7g6 g5e4 d6c7 c1e3
Time: 16109 ms
ply      nodes  score  pv
  1        131     20  c1e3
  2       4717      5  g5e4 d6c7
  3      10487     30  g5e4 d6c7 c1e3
  4     184442     10  g5e4 d6c7 c1e3 c8b7
  5     647714     26  c2a4 d6c7 g2d5 e6d5 c1e3
  6    8400851     16  g2d5 d6d5 c1f4 b8a8 f4e5 c8b7
  7   33024284     27  g2e4 c8d7 e4g6 h7g6 g5e4 d6c7 c1e3
Time: 15797 ms
ply      nodes  score  pv
  1        131     20  c1e3
  2       4717      5  g5e4 d6c7
  3      10487     30  g5e4 d6c7 c1e3
  4     184442     10  g5e4 d6c7 c1e3 c8b7
  5     647714     26  c2a4 d6c7 g2d5 e6d5 c1e3
  6    8400851     16  g2d5 d6d5 c1f4 b8a8 f4e5 c8b7
  7   33024284     27  g2e4 c8d7 e4g6 h7g6 g5e4 d6c7 c1e3
Time: 15765 ms

Nodes: 33024284
Best time: 15765 ms
Nodes per second: 2094784 (Score: 8.615)
Opening book missing.
tscp>
```

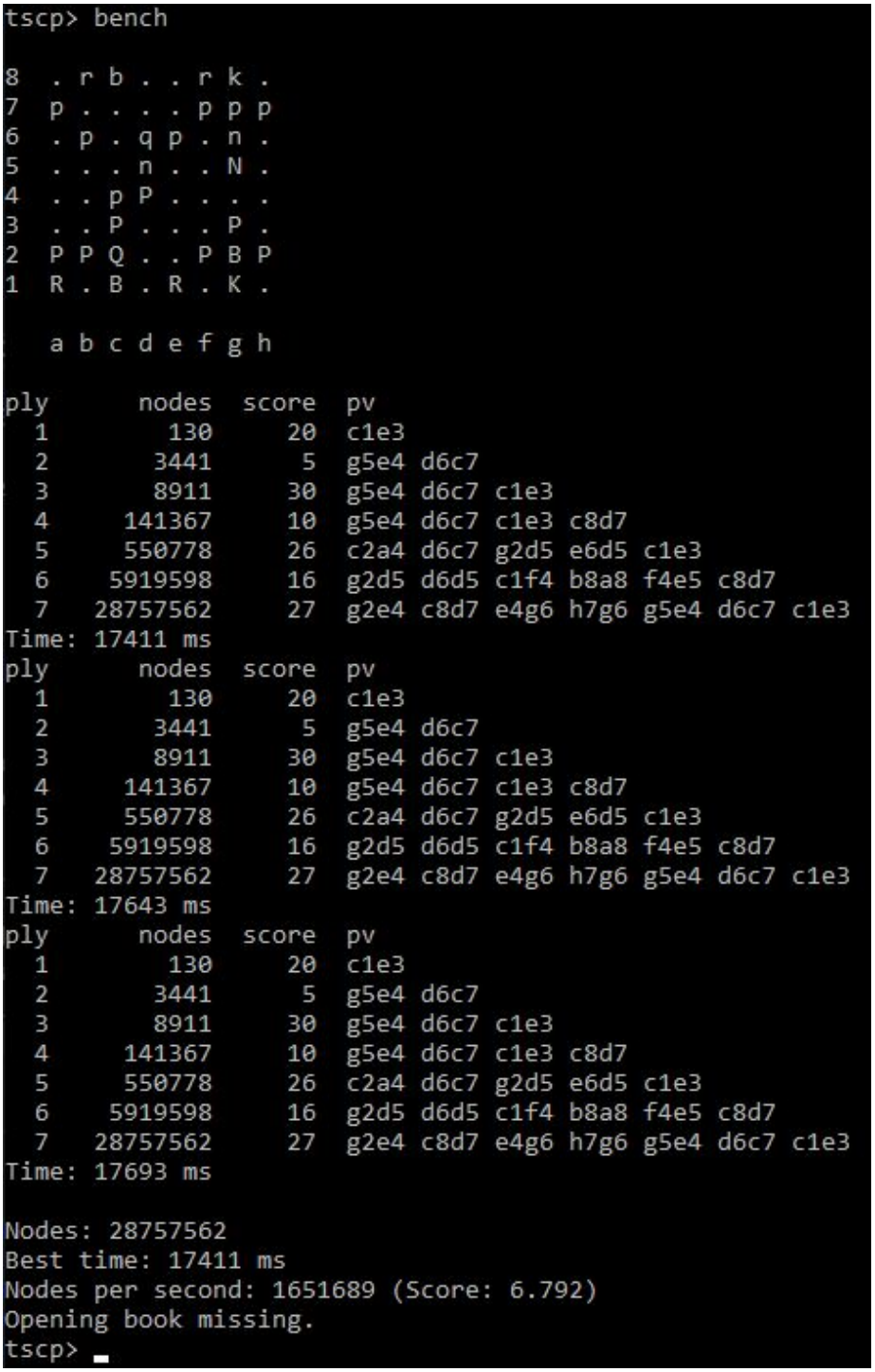
Optimisation 2

Cette optimisation vise à utiliser des tables d'attaque qui régénèrent tous les mouvements possibles pour toutes les pièces. Ainsi les mouvements possibles ne sont pas à générer à chaque déplacement.

Comme nous pouvons le constater, l'optimisation 2 nous a permis de gagner **3.646 secondes** sur les valeurs originales de TSCP. De 21057 ms à 17411 ms.

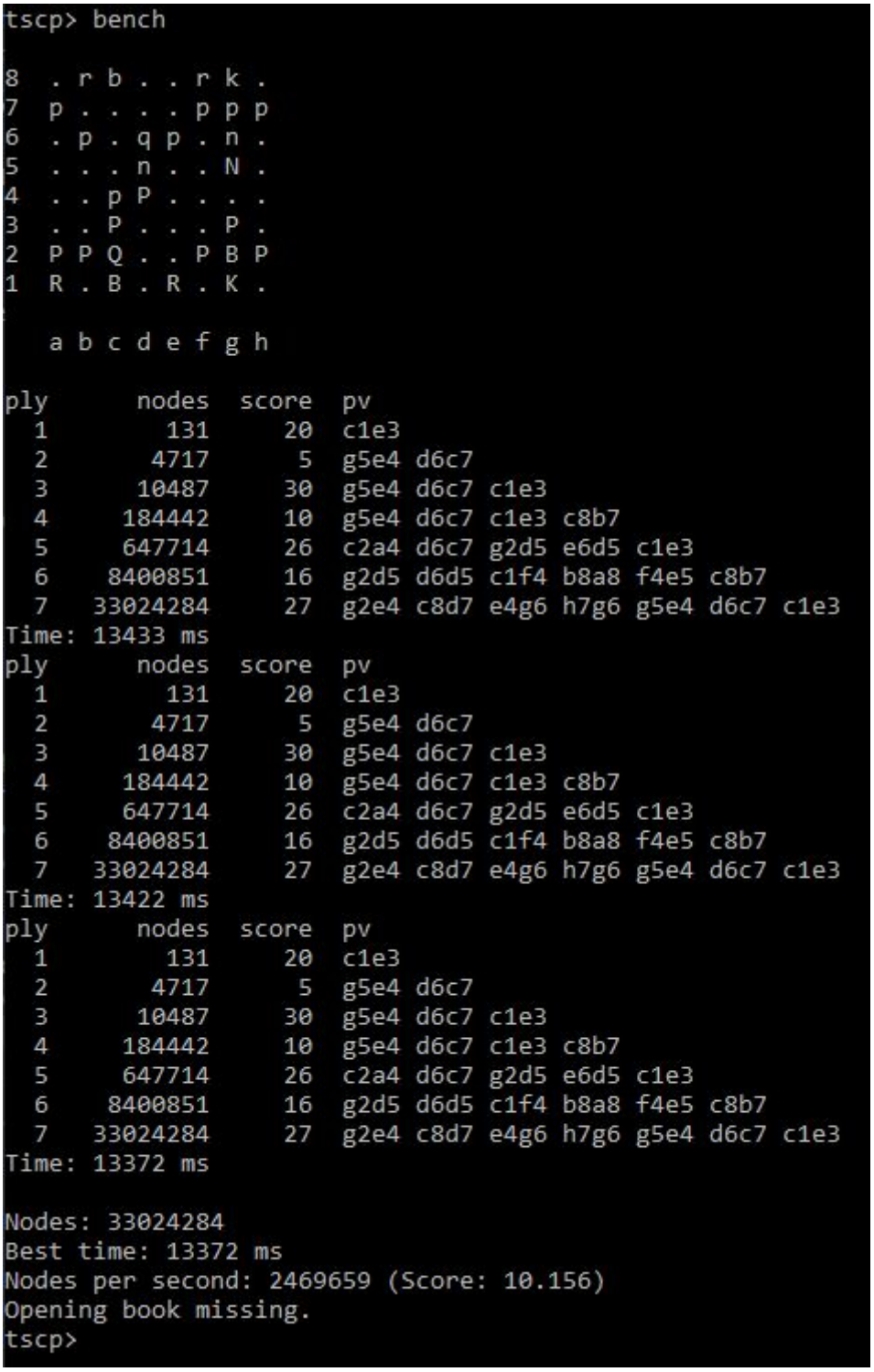
TSCP parcours 1365700 nodes par secondes et l'optimisation 2, 1651689 nodes par secondes.

Nous obtenons donc un score de **6.792** soit environ **+1.2** par rapport à TSCP.



Optimisation 1+2

Finalement, rassembler les optimisations 1 et 2 nous a permis de gagner (environ) **7.6 secondes** sur les valeurs originales de TSCP. De 21057 ms à 17411 ms.
TSCP parcours 1365700 nodes par secondes et les optimisations 1+2, 2469659 nodes par secondes.
Nous obtenons donc un score de **10.156** soit **+4.540** par rapport à TSCP.



Conclusion

Nous pouvons voir qu' à chaque étape d'optimisation, nous avons amélioré les temps d'exécution du programme.
La plus grande amélioration est celle de l'optimisation 1 car elle a permis d'optimiser plusieurs méthodes de parcours en divisant par 4 la taille des boucles.
A contrario, l'optimisation 2 n'a permis d'optimiser que la méthode *attack()* et à donc moins d'impact que la première.