*Actes JIAF 2019*

# Recherche Monte Carlo multi-arbres pour l'exploitation des jeux décomposés

**Aline Hufschmitt** **Jean-Noël Vittaut** **Nicolas Jouandeau**

LIASD - University of Paris 8, France
{alinehuf,jnv,n}@ai.univ-paris8.fr

**Résumé**

Dans cet article nous présentons une variation de la recherche arborescente Monte Carlo (MCTS) consistant à réaliser une recherche dans plusieurs arbres dans le but d'exploiter des jeux décomposés. Cette recherche multi-arbres MCTS (MT-MCTS) consiste à construire simultanément plusieurs arbres de recherche MCTS correspondant aux différents sous-jeux et permet, comme tous les algorithmes de la famille MCTS, d'évaluer les actions en cours de jeu. Nous appliquons MT-MCTS sur des jeux décomposés dans le domaine du *General Game Playing*. Nous présentons des résultats encourageants montrant que cette approche est prometteuse et ouvre de nouvelles pistes de recherche dans le domaine de l'exploitation des décompositions. Des jeux composés complexes sont résolus de 2 fois (*Incredible*) jusqu'à 25 fois plus vite (*Nonogramme*).

**Abstract**

In this paper, we propose a variation of the MCTS framework to perform a search in several trees to exploit game decompositions. Our Multiple Tree MCTS (MT-MCTS) approach builds simultaneously multiple MCTS trees corresponding to the different sub-games and allows, like MCTS algorithms, to evaluate moves while playing. We apply MT-MCTS on decomposed games in the *General Game Playing* framework. We present encouraging results showing that this approach is promising and opens new avenues for further research in the domain of decomposition exploitation. Complex compound games are solved from 2 times faster (*Incredible*) up to 25 times faster (*Nonogramme*).

## 1 Introduction

Le *General Game Playing* (GGP) est une branche de l'Intelligence Artificielle visant à créer des programmes polyvalents capables de jouer à n'importe quel jeu sans intervention humaine. Comme les règles du jeu ne sont pas connues à l'avance, aucun savoir expert ni algorithme spécialisé ne peut être utilisé. Un aspect important du GGP est le développement de méthodes d'analyse automatique des règles dans le but d'accélérer la recherche de la meilleure stratégie.

Parmi les jeux considérés dans le domaine du GGP, certains sont composés de différents sous-jeux indépendants assemblés séquentiellement, ou joués en parallèle de manière synchrone ou asynchrone [3]. Un programme capable d'identifier ces sous-jeux, de trouver la meilleure stratégie pour chacun et de combiner celles-ci, peut significativement réduire le coût de la résolution du jeu global [4]. Plusieurs méthodes ont été proposées pour décomposer des jeux solitaires [5] ou multi-joueurs [18, 7, 8]. Deux approches distinctes ont été proposées pour exploiter ces décompositions, la première inspirée de la planification hiérarchique et la seconde de la vérification de modèle utilisant ASP.

La première approche, nommée *Concept Decomposition Search* [5], est une recherche par approfondissement itératif permettant de résoudre des jeux solitaires. Chaque itération de la recherche est composée de deux étapes : une recherche locale permettant de collecter l'ensemble des plans locaux à profondeur donnée et une recherche globale visant à combiner les sous-plans des différents sous-jeux pour trouver le meilleur plan global. Cette approche est étendue aux jeux multi-joueurs [18] en utilisant des *turn-move sequences* (TMSeqs) comme résultat de la recherche locale. Les TMSeqs indiquent non seulement les actions, mais également les joueurs qui les ont exécutées. La recherche globale est fondée sur les techniques classiques de recherche arborescente, mais utilise les TMSeqs à la place des actions légales, ce qui réduit considérablement la taille de l'arbre par rapport à une recherche standard.

La seconde approche [3] utilise l'*Answer Set Programming* (ASP) pour résoudre les jeux décomposés. Les plans locaux sont combinés à mesure de leur calcul pour trouver

un plan global. Cette recherche est utilisée uniquement sur des jeux solitaires. La manière dont cette approche pourrait être généralisée aux jeux multi-joueurs n'est pas évidente et reste un problème ouvert.

Dans toutes ces approches précédentes, la recherche retourne un plan global ou rien. Dans le cadre du GGP, un temps limité est alloué pour analyser les règles avant le début du jeu (*startclock*). Un algorithme de recherche doit permettre de déterminer rapidement la première action à jouer puis de raffiner la stratégie pendant le temps de réflexion accordé entre chaque coup (*playclock*). Les recherches locales et globales doivent donc être menées en parallèle et permettre d'améliorer le plan tout en jouant.

Dans cet article, nous proposons une nouvelle approche pour résoudre les jeux décomposés fondée sur la recherche arborescente Monte Carlo (MCTS). MCTS constitue l'état de l'art pour les joueurs programmés dans le domaine du GGP, mais aussi pour des joueurs spécialisés comme Alpha Go [16], pour les jeux vidéos et d'autres domaines en dehors des jeux [2]. Notre recherche multi-arbres (MT-MCTS) construit simultanément plusieurs arbres correspondant aux différents sous-jeux. Plutôt que de produire un plan global, MT-MCTS évalue les actions à chaque étape du jeu. Un joueur programmé utilisant MT-MCTS peut donc commencer à jouer tout en explorant la suite du jeu à la recherche la meilleure action suivante. Nous comparons la performance de notre algorithme MT-MCTS, sur des jeux solitaires, avec celle d'une recherche MCTS utilisant la politique *Upper Confidence Bound applied to Trees* (UCT) et les tables de transposition.

Le reste de cet article est organisé comme suit. Dans la section suivante, nous présentons brièvement la recherche MCTS, la politique UCT et ses optimisations courantes. Dans la section 3, nous présentons l'approche MT-MCTS utilisant plusieurs arbres pour résoudre les jeux décomposés. Nous présentons les résultats expérimentaux sur différents jeux solitaires dans la section 4. Dans la section 5, nous discutons ces résultats, nous présentons les défis soulevés par la recherche simultanée dans plusieurs arbres, les possibles extensions de notre algorithme et les problèmes ouverts. Nous concluons dans la section 6.

## 2 MCTS et UCT

Un jeu peut être représenté sous forme d'un arbre. Chaque nœud représente un état du jeu et chaque arc représente un mouvement joint des joueurs [1]. Les algorithmes de la famille MCTS permettent de construire un arbre du jeu de manière incrémentale et asymétrique [2].

MCTS démarre à partir d'un unique nœud représentant l'état courant du jeu et répète les quatre étapes suivantes pendant un nombre d'itérations ou un laps de temps donné : la sélection d'un chemin dans l'arbre du jeu selon la *politique de l'arbre* ; l'expansion de l'arbre avec la création d'un nouveau nœud ; une simulation (*playout*) d'après la *politique par défaut* ; la rétro-propagation du résultat du *playout* pour mettre à jour l'évaluation statistique des nœuds (nombre de visites et cumul des récompenses). Le nombre de *playouts* dans MCTS est un facteur clef pour la qualité de l'évaluation des nœuds de l'arbre [10].

La *politique par défaut* consiste généralement à jouer des coups au hasard jusqu'à atteindre un état terminal du jeu. Pour la sélection, UCT [1] est la *politique de l'arbre* la plus courante. Elle permet un compromis entre l'exploitation des meilleurs coups et l'exploration de l'arbre complet. Dans chaque état visité, le coup choisi est celui qui maximise $U$ :

$$U = \frac{w_i}{n_i} + C * \sqrt{\frac{\log N}{n_i}} \quad (1)$$

avec $w_i$ le cumul des récompenses [2] obtenues pendant les playouts en choisissant le coup $i$, $n_i$ le nombre de visites du nœud enfant correspondant, $N$ le nombre de visites du nœud courant et $C$ une constante équilibrant les termes exploration et exploitation [3].

Une optimisation courante consiste à utiliser une table de transposition : les états identiques du jeu sont représentés par un nœud unique dans l'arbre. Dans le cadre du GGP, pour garantir que tous les jeux se terminent en un nombre fini d'étapes, beaucoup de jeux qui comprennent des cycles utilisent un compteur de coups nommé *stepper*. Les états identiques du jeu, présents à différentes profondeurs dans l'arbre, sont ainsi différenciés par ce *stepper* et les transpositions peuvent uniquement apparaître à une même profondeur. Quand un programme utilise les transpositions, l'évaluation des différentes actions légales est généralement stockée dans les arcs sortants d'un nœud plutôt que dans ses nœuds enfants [15]. Le nombre de visites reste stocké dans le nœud parent.

Une autre optimisation commune, utilisée pour guider la recherche, est l'élagage des branches totalement explorées [12, 17]. Durant l'étape de sélection, plutôt que de retourner dans les branches qui ont été complètement évaluées, le score moyen de la branche est calculé et utilisé pendant l'étape de rétro-propagation.

## 3 Multiple Tree MCTS (MT-MCTS)

Dans le cadre du GGP, l'état du jeu est décrit par un ensemble fini de fluents instanciés dont certains sont vrais.

1. Dans le cadre du GGP, les joueurs jouent toujours simultanément. Pour simuler un jeu à coups alternés, les joueurs qui n'ont pas la main disposent d'un seul coup légal consistant à passer leur tour.

2. Dans le framework GGP, les récompenses sont représentées par un entier entre 0 et 100. Dans cet article, nous considérons que cette récompense est normalisée entre 0.00 et 1.00

3. Dans le framework GGP, $C$ est généralement fixé à une valeur de 0.4 apportant un bon équilibre entre exploration et exploitation dans une majorité de jeux GGP.

Un ensemble de $F$ fluents permet de décrire $2^F$ états distincts. Le résultat de la décomposition est l'identification de différents sous-ensembles de ces fluents représentant l'état des *sous-jeux* [6] que nous nommons ici des *sous-états*. Un jeu peut être décomposé en sous-jeux disjoints [8] dans ce cas ces sous-ensembles de fluents sont disjoints et l'ensemble des sous-jeux correspond à un partitionnement des fluents du jeu global. Une décomposition d'un jeu en sous-jeux non-disjoints peut également être envisagée. Dans ce cas les sous-ensembles de fluents peuvent se recouper e.g. quand les fluents représentent les cases d'un plateau de jeu et chaque sous-jeu représente une ligne ou une colonne du jeu global. La manière d'obtenir ces décompositions est hors du cadre du présent article.

Dans cette section, nous présentons les difficultés résultant de la décomposition, le principe général de notre approche multi-arbres MCTS qui permet de résoudre ces difficultés et des aspects de MT-MCTS pour lesquels nous avons empiriquement développé une politique spécifique : la sélection locale des meilleures actions dans un sous-jeu, la sélection d'une action au niveau global d'après les recommandations des sous-jeux et le marquage des branches totalement explorées.

### 3.1 Difficultés résultant de la décomposition

La décomposition d'un jeu en différents sous-jeux produit des sous-états dans lesquels les actions légales dépendent de la combinaison de ces sous-états, ce qui amène une première difficulté concernant le calcul des coups légaux.

Une autre difficulté apparaît quand un *stepper* est séparé du reste d'un jeu : des cycles peuvent apparaître dans certains sous-jeux. Dans les transpositions d'un sous-état, l'évaluation des actions peut différer en fonction de la profondeur du jeu. Ce problème est évoqué sous les termes de *graph history interaction problem* [13]. Il existe une solution générale pour les jeux à score binaire [9]. Dans le cadre du GGP, les scores sont plus progressifs et cette solution générale n'est donc pas applicable.

La décomposition soulève également un problème pour l'identification des sous-états terminaux. Par exemple, le jeu *Incredible* est décomposé en un labyrinthe (*Maze*), un jeu de cubes (*Blocks World*), un *stepper* et un ensemble d'actions inutiles de contemplation. Le jeu est terminé si le *stepper* atteint 20 ou si le jeu *Maze* est terminé. Dans *Blocks World*, un sous-état n'est jamais terminal par lui-même. Il est possible également d'imaginer un jeu dans lequel deux sous-états seraient tous deux non-terminaux mais dont la combinaison serait terminale et devrait être évitée dans un plan global.

La décomposition amène aussi une difficulté pour l'évaluation des sous-états dont le score peut dépendre d'une combinaison opportune avec d'autres sous-états. De plus, dans le cadre du GGP, le score décrit par le prédicat `goal` est fiable uniquement si l'état courant est terminal [11]. Ces deux faits rendent la fonction d'évaluation moins fiable dans les sous-arbres.

Finalement, le problème de la fiabilité de la décomposition se pose. Si la décomposition est inconsistante, l'évaluation des coups légaux peut être erronée, amenant le joueur programmé à choisir des coups illégaux et à calculer des états incohérents.

Pour éviter tous ces problèmes, nous proposons l'approche suivante : réaliser les simulations dans le jeu global et construire un sous-arbre pour chaque sous-jeu. Les coups légaux, l'état suivant et la fin de jeu peuvent être évalués pour le jeu global dans lequel le score réel est connu. La sélection des actions est réalisée en fonction de l'évaluation des sous-états dans les sous-arbres. Une décomposition inconsistante peut être détectée si durant deux simulations, la même action à partir du même sous-état amène à différents sous-états suivants. Une décomposition partielle [4] mais consistante permet de jouer dans le respect des règles même si son exploitation peut s'avérer moins efficace. Pour prendre en charge les transpositions tout en évitant le *graph history interaction problem*, la version courante de MT-MCTS ne considère les transpositions qu'à la même profondeur dans les sous-arbres. Chaque sous-jeu est donc représenté par un graphe acyclique dirigé avec une racine unique.

### 3.2 Simulations globales et construction des sous-arbres

Notre MT-MCTS itère les quatre même étapes que MCTS, à la différence que l'étape de sélection alterne différentes phases de sélection locale et globale. La figure 1 représente ces quatre étapes. Prenons le jeu solitaire représenté par l'arbre partiel en haut à gauche de la figure. Considérons que la décomposition de ce jeu aboutit à l'identification de deux sous-jeux parallèles asynchrones. Remarquez que les actions `a` et `c` à partir de la racine ont le même effet sur le premier sous-jeu et amènent toutes deux au même sous-état '+'. De même, les actions `b` et `c` ont le même effet sur le second sous-jeu et amènent au même sous-état '`o`'.

Considérons que les actions `a` et `b` ont été testées chacune une fois. Les sous-arbres des sous-jeux 1 et 2 contiennent les arcs représentant ces actions et les nœuds représentant les sous-états atteints. Les transitions ont été évaluées grâce aux scores remontés par les deux *playouts*. Durant la troisième descente (voir 1a), l'action `c`, non testée, est déjà évaluée : cette action amène à la combinaison des sous-états '+' et '`o`' qui ont déjà été ajoutés dans les sous-arbres et évalués. Les transitions représentées par

4. Une décomposition est *partielle* si un sous-jeu peut être décomposé.

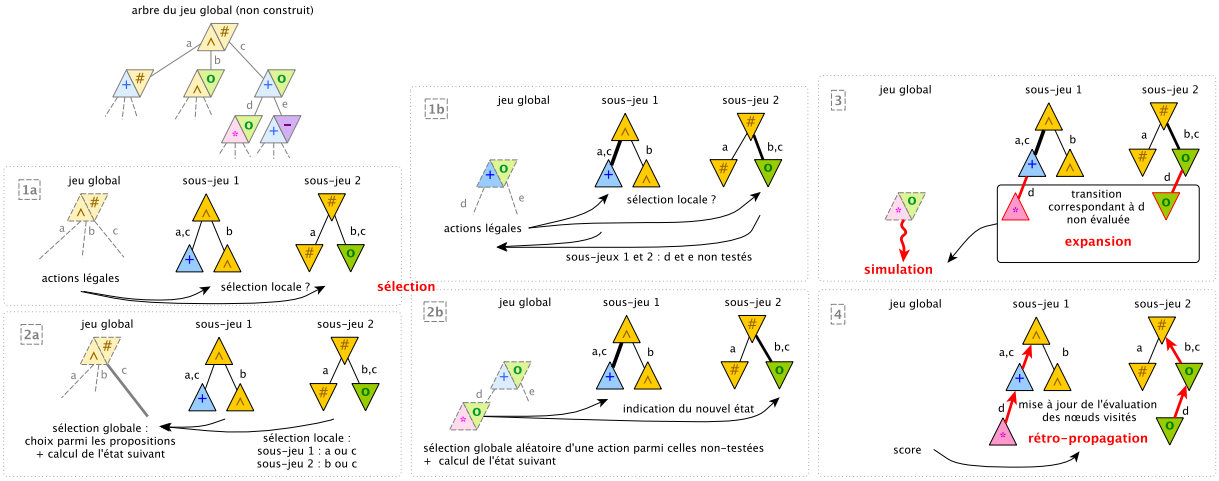


FIGURE 1 – Les quatre étapes de MT-MCTS. Les sous-états identiques sont représentés par un triangle de même orientation, symbole et couleur. La partie de chaque état appartenant au premier sous-jeu est représentée par un triangle pointe en haut, tandis que celle appartenant au second sous-jeu est représentée par un triangle pointe en bas. *Sélection* : 1a Sélection locale dans les sous-jeux, 2a sélection globale en fonction de la sélection locale, 1b sélection locale avec actions non testées, 2b sélection globale aléatoire des actions non testées. *Expansion et simulation* : 3 expansion des sous-arbres et simulation globale. *Rétro-propagation* : 4 rétro-propagation du score global dans chaque sous-jeu.

les arcs de l'arbre, sont étiquetées avec les différentes actions menant au même sous-état : ces dernières forment une *meta-action* [7]. Jouer avec des jeux décomposés permet donc de réduire significativement le facteur de branchement et la taille de l'espace de recherche. Remarquez que les actions b dans le sous-jeu 1 et a dans le sous-jeu 2 ne modifient pas l'état initial du jeu : les transitions qui laissent l'état d'un sous-jeu inchangé permettent d'évaluer l'intérêt de jouer ailleurs que dans ce sous-jeu. Comme les actions légales a, b et c ont été testées une fois, la sélection est effectuée d'après l'évaluation des sous-jeux.

Supposons (voir 2a) que le sous-jeu 1 recommande la transition vers le sous-état '+' (action a ou c) et que le sous-jeu 2 recommande la transition vers le sous-état 'o' (action b ou c). Maintenant, la sélection globale doit faire un choix en fonction de ces recommandations. Ici, l'action c est la plus recommandée et est choisie. Les étapes 1a et 2a sont itérées pour réaliser la descente dans l'arbre global. L'état suivant et les coups légaux sont calculés pour le jeu global ce qui garantit que seules des actions légales sont jouées et que la cohérence de l'état du jeu est préservée.

Quand le sous-état atteint comprend des actions inconnues (voir 1b), la sélection locale retourne la liste des actions non-testées. Une action est sélectionnée au hasard parmi elles (voir 2b), l'état suivant est évalué et les sous-jeux sont interrogés pour savoir si cet état suivant correspond à une transition déjà connue. Si les sous-états atteints sont connus de tous les sous-jeux, l'action est associée aux transitions correspondantes et l'étape 2b est ré-itérée en sélectionnant au hasard une autre action non-testée. S'il ne reste plus d'action non-testée, la descente continue comme décrit à l'étape 1a.

Si le sous-état atteint est nouveau pour au moins un des sous-jeux, une expansion est effectuée (voir 3) pour ajouter la nouvelle transition et le nouveau nœud correspondant dans chaque sous-arbre. Si une transposition existe, le sous-état correspondant est associé à la transition. Un *playout* est exécuté au niveau global à partir de ce nouvel état du jeu.

Dans l'état terminal atteint, le score est évalué (voir 4) puis transmis aux sous-jeux. Chaque sous-jeu remonte le score dans son sous-arbre pour évaluer les transitions et les nœuds visités. L'arbre du jeu global n'est pas construit. Seul l'état courant est conservé en mémoire pour permettre, à chaque étape, d'évaluer les coups légaux, l'état suivant et éventuellement de réaliser un *playout*.

### 3.3 Sélection locale

A chaque étape de la sélection par MT-MCTS, un sous-jeu reçoit un ensemble d'actions légales à évaluer. Le sous-état courant est associé à un nombre de visites $N$. Chaque transition $i$ à partir de ce sous-état est évaluée par un nombre de visites $n_i$ et un score cumulé $w_i$. La sélection locale retourne un ensemble d'actions s'il existe différentes transitions avec la même évaluation $U$ ou si la meilleure transition est étiquetée avec différentes actions.

Nous avons exploré différentes approches pour réaliser la sélection locale. La première est une application standard de la politique UCT. Cependant, cette approche n'est pas satisfaisante car une transition dans un sous-jeu peut alors recevoir une bonne évaluation sans avoir contribué au score global : l'évaluation a été obtenue grâce à une séquence d'actions menant à une évaluation positive dans un autre sous-jeu. Un autre problème plus grave survient dans le cas des jeux à score binaire : le score est toujours zéro jusqu'à ce qu'une solution soit trouvée. Dans ce cas, la recherche se ramène à une exploration en largeur d'abord et ne permet pas la découverte des bonnes combinaisons de sous-états.

Dans le cadre du GGP, un état du jeu est décrit par un ensemble fini de fluents instanciés dont certains sont vrais. La décomposition partitionne ces fluents en différents groupes qui représentent les états des sous-jeux. Dans un état terminal global, il est possible de garder uniquement les fluents correspondant à l'un des sous-jeux, de donner la valeur *indéfini* aux autres fluents et d'évaluer les règles logiques du jeu avec une logique tri-valuée. Les prédicats `goal` vrais ou indéfinis représentent les scores possibles d'après l'état de ce sous-jeu. Le `goal` de score maximum (*lmax*) correspond au score maximum potentiel qui peut être obtenu si le reste de l'état du jeu correspond à la meilleure configuration possible dans les autres sous-jeux.

Le score *lmax* est indicatif, le vrai score maximum peut varier légèrement car une logique tri-valuée ne garantit pas l'information la plus précise [14]. L'évaluation *lmax* est néanmoins une estimation précieuse de la valeur d'un sous-état. Cette information peut être rétro-propagée en plus du score global et cumulée dans une variable $w_i^{max}$. Pour une transition donnée, $w_i/n_i$ est un estimateur du score global et $w_i^{max}/n_i$ est un estimateur du score local. Ces deux informations peuvent être utilisées dans une politique dérivée d'UCT (voir eq.1). Les actions choisies par le sous-jeu sont celles associées aux transitions dont l'évaluation $U$ est maximum :

$$U = \alpha \frac{w_i}{n_i} + (1-\alpha)\frac{w_i^{max}}{n_i} + C * \sqrt{\frac{\log N}{n_i}} \qquad (2)$$

avec $\alpha \in [0, 1]$ pondérant les estimateurs de score global et local.

Pour éviter de retourner dans les branches déjà complètement explorées, les transitions correspondant à ces branches sont exclues de la sélection locale aussi longtemps que des transitions non totalement explorées existent.

### 3.4 Sélection globale

Dans un jeu comme *Incredible*, le sous-jeu *Maze* est rapidement complètement exploré. Ce sous-jeu recommande alors systématiquement la séquence d'actions menant au gain maximum dans ce sous-jeu (45% du score total). Cependant, terminer ce sous-jeu met fin prématurément au jeu global. Pour un algorithme fondé sur une balance entre exploration et exploitation, il faut un très grand nombre de visites de l'action terminale de *Maze* pour orienter la recherche vers l'exploration des autres actions possibles (jouer dans le sous-jeu *Blocks World*). Pour éviter ce problème, les coups légaux sont étiquetés en fonction de leur statut *terminal* ou non. Si certaines actions sont terminales, le score correspondant est évalué. Si une action terminale retourne le score maximum possible pour le joueur courant, elle est directement sélectionnée. Dans le cas contraire, la sélection locale est appliquée sur les actions légales non-terminales.

La sélection globale de la meilleure action est réalisée d'après les actions recommandées par les sous-jeux. Dans les jeux en série ou les jeux parallèles synchrones, l'intersection des ensembles d'actions recommandées est toujours non-vide. La sélection globale est alors évidente : une action est choisie au hasard dans cette intersection. En revanche, dans un jeu parallèle asynchrone, différents sous-jeux peuvent proposer des ensembles d'actions disjoints. Il est alors nécessaire de définir une politique pour le choix de l'action au niveau global. Pour définir une politique compatible avec tous les jeux, nous proposons une politique de vote. Chaque sous-jeu recommandant une action lui apporte une voix. Dans le cas des jeux en série ou parallèles synchrones, la meilleure action obtient autant de voix qu'il y a de sous-jeux. Dans le cas des jeux parallèles asynchrones, il est possible que chaque action n'obtienne qu'une seule voix.

Parmi les actions ayant reçu le plus de voix, celles apportant la plus grande espérance de gain sont sélectionnées pour diriger la recherche sur les actions menant à la meilleure combinaison de sous-états. Quand le but du jeu correspond à la conjonction des buts des sous-jeux, ce gain maximum espéré pour une action $m$ parmi $T$ sous-jeux est le produit des probabilités de gain dans chaque sous-jeu $s$ :

$$\prod_{s=1}^{T} \frac{w_m^s}{n_m^s} \qquad (3)$$

Quand le but du jeu correspond à la disjonction des buts des sous-jeux, ce gain maximum espéré est la somme des probabilités de gain dans chaque sous-jeu $s$ :

$$\sum_{s=1}^{T} \frac{w_m^s}{n_m^s} \qquad (4)$$

$w_m^s$ est le cumul des récompenses gagnées durant les *playouts* et $n_m^s$ le nombre de visites associés à la transition étiquetée par cette action $m$ dans le sous-jeu $s$. Si plusieurs actions offrent la même espérance de gain maximum, une d'elles est sélectionnée aléatoirement.

### 3.5 Marquage des branches totalement explorées

Différentes séquences d'actions peuvent mener à visiter de nombreuses fois la même branche d'un sous-arbre. Pour éviter de re-visiter inutilement les branches totalement explorées, dans le cas d'UCT, il est courant de les étiqueter pour stopper la descente ou encourager la visite des branches voisines. Cependant, dans le cas des sous-jeux, un sous-état n'est pas toujours terminal en fonction du reste de l'état global. Il est donc nécessaire de développer une approche spécifique pour étiqueter les branches totalement explorées dans les sous-jeux. Nous proposons de résoudre ce problème par une simple révision du marquage durant les descentes successives.

Durant l'étape de sélection, la liste des coups légaux est calculée. Nous vérifions à cette occasion si les sous-états suivants connus comme terminaux sont bien terminaux dans la situation courante. Dans le cas contraire, l'étiquetage est révisé et cette révision est rétro-propagée le long du chemin visité pendant la descente. Lorsque toutes les transitions à partir d'un sous-état sont signalées comme totalement explorées, la transition menant à ce sous-état est également étiquetée comme "totalement explorée". Dans le cas de MT-MCTS, ce marquage est utilisé non pas pour stopper définitivement la recherche mais pour encourager la visite des branches voisines en priorité.

## 4 Expérimentations

Nous présentons ici des expérimentations menées avec MT-MCTS [5]. Premièrement nous évaluons différentes pondérations de notre politique de sélection locale et deuxièmement nous comparons la performance de MT-MCTS face à UCT. Nous montrons que notre approche peut réduire significativement le nombre de simulations et le temps de résolution des jeux. Nous menons nos expérimentations sur différents jeux solitaires : *Incredible*, différentes grilles de *Nonogramme* de taille $5 \times 5$ et $6 \times 6$ (fig.2) et *Queens08lg*.

*Incredible* est un jeu intéressant car il est possible de mettre prématurément fin au jeu avec un score sous-optimal. Il est couramment utilisé pour évaluer les joueurs programmés capables d'exploiter les décompositions. Le joueur met fin à la partie et obtient 45 points s'il rapporte un trésor à l'entrée du labyrinthe (*Maze*), additionné à $25 + 30$ points s'il réalise au préalable deux piles de cubes (*Blocks World*).

Les *Nonogrammes* sont des puzzles logiques dans lesquels les cases d'une grille doivent être colorées ou laissées blanches en fonction de nombres placés en bout des colonnes et des lignes. Le score est binaire et UCT n'apporte pas d'amélioration par rapport à une recherche en profondeur ou en largeur d'abord sur ce jeu.

*Queens08lg* est au contraire rapidement résolu par UCT. Il s'agit d'un *puzzle des huit reines* dans lequel il est illégal de placer une reine dans une position où elle pourrait capturer une autre reine en un coup. Le jeu est terminé quand aucune reine ne peut être ajoutée sur l'échiquier ou si le joueur déclare forfait. Le joueur reçoit une partie des points pour chacune des huit reines à placer.

Nous avons décomposé ces jeux en utilisant une méthode de décomposition statistique [8]. Pour chaque jeu et chaque configuration, nous avons réalisé 10 tests et nous mesurons le nombre moyen de *playouts* et le temps moyen nécessaires pour résoudre le jeu. Un jeu est considéré résolu quand une feuille de score maximum est trouvée.

### 4.1 Pondération de la politique de sélection locale

Le but de notre première expérimentation est de comparer différentes valeurs de $\alpha$ dans la politique de sélection locale (eq.2). Nous utilisons $C = 0.4$ qui permet un bon équilibre entre exploration et exploitation dans une majorité de jeux GGP. Nous présentons les résultats de nos tests sur deux jeux : *Incredible* et *Nonogramme "damier"*. Les résultats sont présentés sur la figure 3.

En utilisant uniquement l'estimateur de score global ($\alpha = 1$) dans la politique de sélection locale, il n'est pas possible de résoudre le *Nonogramme* à cause de son score binaire. Le score estimé est toujours zéro et les actions choisies au hasard ont très peu de chance de mener à la bonne combinaison de sous-états. Au contraire, l'utilisation exclusive de l'estimateur de score local ($\alpha = 0$) permet de guider la recherche dans les sous-arbres et de résoudre le *Nonogramme* en moins de 5 secondes. Cependant, l'utilisation de $\alpha = 1$ donne de meilleurs résultats dans *Incredible* tandis que $\alpha = 0$ nécessite presque deux fois plus de temps pour résoudre le jeu. En faisant varier $\alpha$, nous remarquons qu'une faible participation de l'estimateur de score local ($\alpha = 0.75$) permet d'obtenir un résultat encore meilleur dans *Incredible* et une faible participation de l'estimateur de score global ($\alpha = 0.25$) permet d'obtenir la résolution la plus rapide du *Nonogramme*. Cependant, l'importance des écarts types, du même ordre de grandeur que le temps de résolution ou un ordre en dessous, ne permettent pas d'identifier un effet vraiment significatif de la variation de la pondération. En considérant l'écart type, les temps de résolution sont similaires dans les trois tests mêlant les estimateurs de score local et global. Plus d'expérimentation seront nécessaires pour vérifier l'influence d'une pondération inégale de ces deux informations.

Toutefois, l'association de ces deux estimateurs semble souhaitable pour constituer une politique polyvalente. Dans les expérimentations suivantes, nous utilisons la pondération $\alpha = 0.5$ dans la politique de sélection locale car elle

5. Les expérimentations ont été réalisées sur un cœur d'un Intel Core i7 2,7GHz avec 8Go de DDR3 à 1.6GHz.

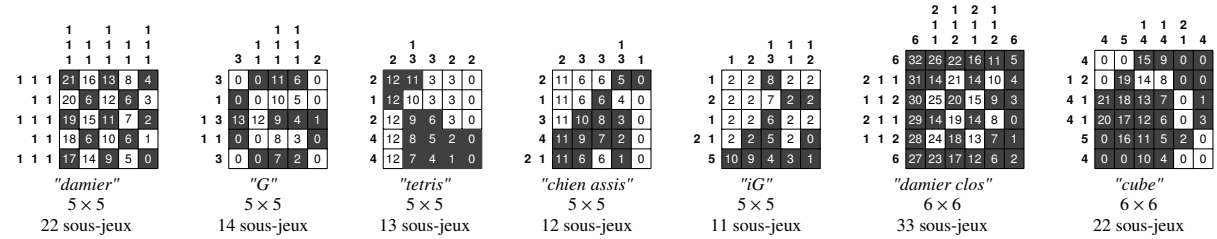


FIGURE 2 – Différentes grilles de Nonogramme (résolues). Le numéro à l'intérieur de chaque case indique le sous-jeu auquel elle appartient. Chaque case dont le statut peut être déterminé indépendamment des autres constitue un sous-jeu.

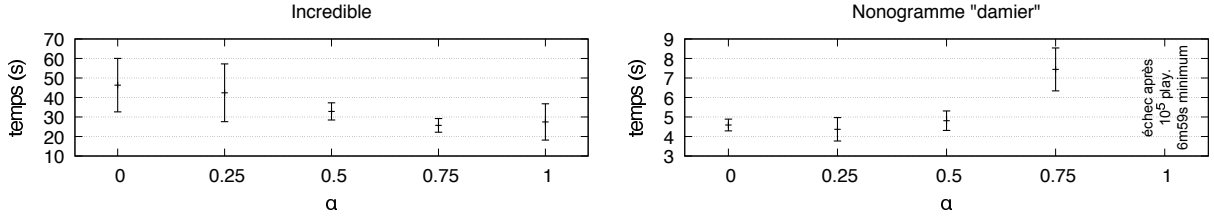


FIGURE 3 – Temps moyen sur 10 tests pour résoudre *Incredible* et *Nonogramme "damier"* avec différentes valeurs de $\alpha$ pour la politique de sélection locale de MT-MCTS (eq.2).

donne globalement le meilleur résultat.

### 4.2 MT-MCTS vs. UCT

Dans une seconde expérimentation (fig.4), nous comparons l'efficacité de MT-MCTS face à UCT en termes de nombre de *playouts* et de temps utilisés pour la résolution des jeux.

MT-MCTS est significativement meilleur que UCT sur le jeu *Incredible* en terme de *playouts* nécessaires pour résoudre le jeu. MT-MCTS utilise vingt fois moins de *playouts* mais la sélection des actions est significativement plus longue. Au final, le jeu est résolu deux fois plus vite.

La comparaison de nos résultats avec ceux obtenus avec la *Concept Decomposition Search* (*Fluxplayer*) [5] ou en encodant le problème en ASP [3] est difficile car ces approches sont totalement différentes de UCT. *Fluxplayer* met 2 heures pour résoudre *Incredible* en calculant 41 millions d'états (à comparer aux 280 milles *playouts* pour UCT [6]). L'exploitation du jeu décomposé réduit ce temps à 45 secondes et 3212 états. Ce temps de résolution est supérieur au nôtre bien que moins d'états soient calculés par leur approche. L'encodage de *Incredible* en ASP permet la résolution du jeu en 6.11 secondes. Ce temps est réduit à 1.94 secondes en exploitant la décomposition, soit un facteur de 3. Il faut noter que cette approche est optimisée pour les jeux solitaires. Comme notre approche requiert 21 fois moins de *playouts* pour la résolution du jeu décomposé, en optimisant l'étape de sélection de MT-MCTS, nous espérons obtenir une performance similaire voire meilleure.

6. Comme chaque *playout* donne lieu à une expansion de l'arbre, nous pouvons comparer le nombre de *playouts* avec le nombre d'états calculés.

Pour un puzzle simple comme *Queens08lg*, même si la résolution du jeu décomposé nécessite 3 fois moins de *playouts*, le temps de décomposition est trop important comparé au gain qui peut être espéré dans le temps de résolution.

Sur les grilles de *Nonogramme* (5×5 et 6×6), MT-MCTS est significativement meilleur que UCT. La résolution est 25 fois plus rapide pour *"tetris"*. Ce gain n'est pas directement lié au nombre de sous-jeux identités : pour *"damier"*, qui a un nombre plus important de sous-jeux, le temps de résolution n'est que 6 fois plus rapide. Le gain observé dans le temps de résolution est directement lié au nombre de simulations nécessaires : de 300 fois moins pour *"iG"* jusqu'à 2400 fois moins pour *"damier clos"* (sans mentionner les 4 tests où UCT a été interrompu après $10^7$ *playouts* sans trouver de solution). L'étape de sélection plus lourde est largement compensée ici par le gain significatif dans le nombre de simulations. Le temps nécessaire pour résoudre *"cube"* avec MT-MCTS est de 2 heures en moyenne. UCT trouve parfois une solution en moins de 2 heures, mais la majorité des tests (8/10) échoue à trouver une solution après 3 heures en moyenne.

## 5 Discussion

Comme certaines combinaisons de sous-états peuvent ne jamais être testées, le partitionnement de la recherche sur plusieurs arbres n'offre en théorie aucune garantie de trouver la solution avec un nombre infini de *playouts*. En pratique, une bonne politique de sélection permet de guider la recherche vers la séquence d'actions adéquate pour at-

| Jeu | UCT / MT-MCTS | # playouts | temps (décomp.) | $\sigma$ | # échec |
|---|---|---|---|---|---|
| *Incredible* | UCT | 280158 | 1m | 14.3s | - |
| | MT-MCTS | 13199 | 32.86s (2.50s) | 4.4s | - |
| *Queens08lg* | UCT | 67 | 0.01s | <0.1s | - |
| | MT-MCTS | 22 | 1.30s (1.29s) | <0.1s | - |
| *Nonogramme "damier"* | UCT | 432019 | 31.31s | 16.5s | - |
| | MT-MCTS | 776 | 4.81s (0.69s) | 0.5s | - |
| *Nonogramne "G"* | UCT | 2988921 | 4m24s | 4m8s | - |
| | MT-MCTS | 9344 | 36.61s (0.77s) | 16.45s | - |
| *Nonogramne "tetris"* | UCT | 4969111 | 7m20s | 3m53s | 1 *(après $10^7$ play. = 15m17s)* |
| | MT-MCTS | 3767 | 16.36s (1.01s) | 6.5s | - |
| *Nonogramne "chien assis"* | UCT | 2552082 | 3m43s | 2m50s | - |
| | MT-MCTS | 5476 | 26.27s (0.98s) | 14.92s | - |
| *Nonogramne "iG"* | UCT | 3086172 | 4m34s | 3m9s | - |
| | MT-MCTS | 10232 | 28.60s (0.85s) | 12.40s | - |
| *Nonogramne "damier clos"* | UCT | 4284872 | 13m7s | 6m41s | 4 *(après $10^7$ play. = 30m25s minimum)* |
| | MT-MCTS | 1762 | 49.26s (2.40s) | 3.72s | - |
| *Nonogramne "cube"* | UCT | 21438731 | 1h17m23s | 19m16s | 8 *(après $5 \times 10^7$ play. = 3h57s minimum)* |
| | MT-MCTS | 358608 | 1h53m8s (2.75s) | 45m7s | - |

FIGURE 4 – Comparaison des temps de résolution des différents jeux avec UCT et MT-MCTS. Les colonnes présentent le nombre moyen de *playouts* et le temps moyen pour résoudre les puzzles (échecs exclus) sur 10 tests. La partie du temps utilisé pour la décomposition est présentée entre parenthèses. La colonne $\sigma$ indique l'écart type. La colonne "échec" indique le nombre de recherches stoppées avant d'avoir trouvé une solution avec, entre parenthèses, le nombre de playouts exécutés et le temps minimum utilisé pour les réalisés.

teindre la bonne combinaison de sous-états.

Le problème, dans le cadre du GGP, est que la politique optimale dépend de la structure du jeu. Par exemple, exclure les branches totalement explorées de la sélection locale peut rapidement guider la recherche vers une solution dans *Incredible* car cela permet d'éviter de ré-explorer le chemin menant à un score sous-optimal. Cependant, cela peut ralentir la résolution d'un jeu comme *Nonogramme* dans lequel jouer les actions déjà identifiées comme bonnes permet de colorer certaines cases et de guider la découverte des bonnes actions suivantes.

Malgré ce ralentissement de la résolution pour les *Nonogrammes*, MT-MCTS est plus efficace que UCT sur ces puzzles. Cependant, comme une intéressante combinaison de sous-états peut rester inexplorée pendant un long moment, nous supposons qu'une valeur fixe de $\alpha$ dans notre politique de sélection locale peut ne pas être aussi efficace sur tous les jeux GGP. Des recherches complémentaires sont donc nécessaires. On peut envisager de nombreuses politiques distinctes qui permettraient de descendre rarement dans les branches totalement explorées et d'améliorer l'étape de sélection de MT-MCTS. Trouver une politique pour MT-MCTS dont l'efficacité serait prouvée sur tous les jeux est un problème ouvert intéressant.

Les contraintes chiffrées indiquées en bout des lignes et colonnes font que *Nonogramme* présente naturellement une composition en lignes et en colonnes. La structure de MT-MCTS permet d'exploiter un jeu décomposé de cette manière. Une autre piste de recherche à considérer est donc l'exploitation de différents sous-jeux qui se chevauchent et, plus généralement, de sous-jeux non disjoints.

Notre version de MT-MCTS ne considère pas toutes les transpositions pour éviter les jeux contenant des cycles. Cinquante cinq pour cent des jeux GGP utilisent un *stepper*. Le développement d'une politique de sélection spécifique pour tirer avantage des transpositions dans les jeux contenant des cycles est donc une piste intéressante qui pourrait significativement améliorer le niveau des joueurs programmés.

## 6 Conclusion

Dans cet article nous avons proposé une extension de MCTS pour effectuer une recherche dans plusieurs arbres représentant les différentes parties d'un problème décomposé. Nous avons testé cette idée sur différents jeux solitaires dans le cadre du *General Game Playing*. Jouer avec des jeux décomposés permet d'espérer un réel changement d'échelle dans leur vitesse de résolution. Nos tests avec une politique de sélection pondérée donnent des résultats prometteurs : les jeux sont résolus de 2 fois plus vite (*Incredible*) jusqu'à 25 fois plus vite (*Nonogramme*). La recherche multi-arbres (MT-MCTS) peut être étendue aux jeux multijoueurs comme les approches MCTS conventionnelles et permet également l'exploitation de sous-jeux non-indépendants.

La nouvelle approche MT-MCTS ouvre différentes pistes de recherche : le développement d'une politique de sélection efficace sur les différents types de jeux composés, la prise en charge du cas particulier des jeux contenant des cycles et utilisant un *stepper*, l'exploitation des sous-jeux se chevauchant et l'exploitation de décompositions incomplètes voire imparfaites.

## Références


[1] Auer, Peter, Nicolò Cesa-Bianchi et Paul Fischer: *Finite-time Analysis of the Multiarmed Bandit Problem*. Mach. Learn., 47(2-3) :235–256, mai 2002, ISSN 0885-6125.

[2] Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis et Simon Colton: *A survey of Monte Carlo Tree Search methods*. Computational Intelligence and AI in Games, IEEE Transactions on, 4(1) :1–43, 2012.

[3] Cerexhe, Timothy, David Rajaratnam, Abdallah Saffidine et Michael Thielscher: *A Systematic Solution to the (De-)Composition Problem in General Game Playing*. Dans *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2014.

[4] Cox, Evan, Eric Schkufza, Ryan Madsen et Michael Genesereth: *Factoring General Games using Propositional Automata*. Dans *Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09)*, pages 13–20, 2009.

[5] Günther, Martin, Stephan Schiffel et Michael Thielscher: *Factoring General Games*. Dans *Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09)*, pages 27–33, 2009.

[6] Hufschmitt, Aline: *Décomposition des jeux dans le domaine du General Game Playing*. Thèse de doctorat, LIASD, Université Paris 8, 2018.

[7] Hufschmitt, Aline, Jean Méhat et Jean Noël Vittaut: *A General Approach of Game Description Decomposition for General Game Playing*. Dans *Proceedings of the IJCAI-16 Workshop on General Game Playing (GIGA'16)*, pages 23–29, juillet 2016.

[8] Hufschmitt, Aline, Jean Noël Vittaut et Nicolas Jouandeau: *Statistical GGP Games Decomposition*. Dans *Proceedings of the IJCAI-18 Workshop on Computer Games (CGW 2018)*, page 19p., juillet 2018.

[9] Kishimoto, Akihiro et Martin Müller: *A General Solution to the Graph History Interaction Problem*. Dans *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 644–649, 2004.

[10] Kocsis, Levente et Csaba Szepesvári: *Bandit Based Monte-Carlo Planning*. Dans *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, 2006.

[11] Love, Nathaniel, Timothy Hinrichs, David Haley, Eric Schkufza et Michael Genesereth: *General Game Playing : Game Description Language Specification*. Rapport technique LG-2006-01, Stanford University, janvier 2006.

[12] Mehat, Jean et Tristan Cazenave: *Combining UCT and nested Monte Carlo search for single-player general game playing*. IEEE Transactions on Computational Intelligence and AI in Games, 2(4) :271–277, 2011.

[13] Palay, A.J.: *Searching With Probabilities*.

[14] Reps, Thomas W., Alexey Loginov et Shmuel Sagiv: *Semantic Minimization of 3-Valued Propositional Formulae*. Dans *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, page 40, 2002.

[15] Saffidine, Abdallah, Jean Méhat et Tristan Cazenave: *UCD : Upper Confidence Bound for Rooted Directed Acyclic Graphs*. Dans *TAAI 2010*, pages 467–473, Piscataway, NJ -, 2010. TAAI 2010, IEEE.

[16] Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel et Demis Hassabis: *Mastering the game of Go without human knowledge*. Nature, 550 :354–359, octobre 2017.

[17] Winands, Mark H., Yngvi Björnsson et Jahn Takeshi Saito: *Monte-Carlo Tree Search Solver*. Dans *Proceedings of the 6th International Conference on Computers and Games*, CG '08, 2008.

[18] Zhao, Dengji, Stephan Schiffel et Michael Thielscher: *Decomposition of Multi-Player Games*. Dans *Proceedings of the Australasian Joint Conference onArtificial Intelligence*, tome 5866, pages 475–484. Springer, 2009.