Multi-objective sequential decision making

Weijia Wang

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE d'Informatique, ED 427
Laboratoire de Recherche en Informatique

Informatique

THÈSE DE DOCTORAT

soutenue le 11/07/2014

par

Weijia WANG

Multi-Objective Sequential Decision Making

Directrice de thèse :	Michèle Sebag	DR CNRS, Université Paris-Sud, LRI/TAO
Co-encadrant de thèse :	Marc Shoenauer	DR INRIA, Université Paris-Sud, LRI/TAO

Composition du jury :

Président du jury :	Dominique Gouyou-Beauchamps	Professeur, Université Paris-Sud, LRI
Rapporteurs :	Jin-Kao Hao	Professeur, Université d'Angers, LERIA
	Philippe Preux	Professeur, Université de Lille 3, INRIA
Examinateurs :	Yann Chevaleyre	Professeur, Université Paris 13, LIPN
	Cécile Germain-Renaud	Professeur, Université Paris-Sud, LRI/TAO
Directrice de thèse :	Michèle Sebag	DR CNRS, Université Paris-Sud, LRI/TAO

Multi-Objective Sequential Decision Making

Ph.D. Thesis

École Doctorale d'Informatique, ED 427

Université Paris-Sud 11

By **Weijia WANG**

Presented and publicly defended :

On July 11 2014

In Orsay, France

With the following jury :

Yann Chevaleyre,	Professor, University of Paris 13, LIPN	(Examiner)
Cécile Germain-Renaud,	Professor, University of Paris-Sud, LRI/TAO	(Examiner)
Dominique Gouyou-Beauchamps,	Professor, University of Paris-Sud, LRI	(Examiner)
Jin-Kao Hao,	Professor, University of Angers, LERIA	(Reviewer)
Philippe Preux,	Professor, University of Lille 3, INRIA Lille	(Reviewer)
Michèle Sebag,	Senior scientist CNRS, University of Paris-Sud, LRI/TAO	(Ph.D Advisor)

Examiners :

Jin-Kao Hao,	Professor, University of Angers, LERIA, France
Philippe Preux,	Professor, University of Lille 3, INRIA Lille, France

La prise de décisions séquentielles multi-objectif

Thèse de doctorat

École Doctorale d'Informatique, ED 427

Université Paris-Sud 11

Par **Weijia WANG**

Présentée et soutenue publiquement

Le 11 juillet 2014

À Orsay, France

Devant le jury ci-dessous :

Yann Chevaleyre,	Professeur, Université Paris 13, LIPN	(Examinateur)
Cécile Germain-Renaud,	Professeur, Université Paris-Sud, LRI/TAO	(Examinateur)
Dominique Gouyou-Beauchamps,	Professeur, Université Paris-Sud, LRI	(Examinateur)
Jin-Kao Hao,	Professeur, Université d'Angers, LERIA	(Rapporteur)
Philippe Preux,	Professeur, Université de Lille 3, INRIA Lille	(Rapporteur)
Michèle Sebag,	DR CNRS, Université Paris-Sud, LRI/TAO	(Directrice de Thèse)

Rapporteurs :

Jin-Kao Hao,	Professeur, Université de Angers, LERIA, France
Philippe Preux,	Professeur, Université de Lille 3, INRIA Lille, France

Abstract

This thesis is concerned with multi-objective sequential decision making (MOSDM).

The motivation is twofold. On the one hand, many decision problems in the domains of e.g., robotics, scheduling or games, involve the optimization of sequences of decisions. On the other hand, many real-world applications are most naturally formulated in terms of multi-objective optimization (MOO).

The proposed approach extends the well-known Monte-Carlo tree search (MCTS) framework to the MOO setting, with the goal of discovering several optimal sequences of decisions through growing a single search tree. The main challenge is to propose a new reward, able to guide the exploration of the tree although the MOO setting does not enforce a total order among solutions.

The main contribution of the thesis is to propose and experimentally study two such rewards, inspired from the MOO literature and assessing a solution with respect to the archive of previous solutions (Pareto archive): the hypervolume indicator and the Pareto dominance reward.

The study shows the complementarity of these two criteria. The hypervolume indicator suffers from its known computational complexity; however the proposed extension thereof provides fine-grained information abut the quality of solutions with respect to the current archive. Quite the contrary, the Pareto-dominance reward is linear but it provides increasingly rare information.

Proofs of principle of the approach are given on artificial problems and challenges, and confirm the merits of the approach. In particular, MOMCTS is able to discover policies lying in non-convex regions of the Pareto front, contrasting with the state of the art: existing Multi-Objective Reinforcement Learning algorithms are based on linear-scalarization and thus fail to sample such non-convex regions.

Finally MOMCTS honorably competes with the state of the art on the 2013 MOPTSP competition.

Résumé en Français (extended abstract in French)

Cette thèse porte sur le problème de la prise de décision séquentielle multi-objectif. Les algorithmes de la prise de décisions séquentielles, plus spéciquement fondés sur la recherche Monte-Carlo arborescente (MCTS) [Kocsis and Szepesvári, 2006], ont été étendus au cas multi-objectif en s'inspirant des indicateurs à base de population proposés dans la littérature de l'optimisation multi-objectif, l'indicateur d'hypervolume et la relation de dominance.

0.1 Contexte / Motivation

0.1.1 Décision séquentielle

La prise de décision compose une partie importante de nos activités quotidiennes. La prise de décision repose généralement sur une mesure d'ordre total (comme la fonction de récompense), indiquant la qualité des décisions à optimiser. Le problème de la prise de décision séquentielle (SDM) est plus complexe en ce sens que les séquences de décision optimales, aussi appelées politiques de décision ou stratégies, ne sont généralement pas formées en sélectionnant la meilleure décision individuelle à chaque étape : les décisions qui composent la séquence optimale ne sont pas indépendantes. Les applications typiques de SDM incluent les jeux [Aliprantis and Chakrabarti, 2000], la programmation [Zhang and Dietterich, 1995], et la robotique [Mahadevan and Connell, 1992].

Une des principales difficultés du problème SDM est la taille de l'espace de recherche, exponentielle en fonction de la longueur des séquences considérées. Le but de l'apprentissage par renforcement (RL), produire des séquences de décisions optimales à l'échelle globale, procède dans le cas général en identifiant la fonction de valeur attachée à un état ou une paire (état,action), i.e. la somme des récompenses que l'on peut espérer recevoir après avoir visité cet état, ou après avoir effectué cette action dans cet état.

0.1.2 Optimisation multi-objectif

Indépendamment, de nombreux problèmes de décision dans le monde réel impliquent de multiples objectifs ; par exemple, un processus de fabrication cherchera souvent à minimiser simultanément le coût et le risque de la production. Ces problèmes sont appelés optimisation multi-objectif (MOO). Pour un problème de MOO non trival, il n'existe pas de solution unique qui optimise simultanément chaque objectif. Les fonctions objectifs sont antagonistes. Deux solutions ne sont pas nécessairement comparables ; par exemple, un plan de production pourrait être d'un coût élevé et à faible risque, et un autre de faible coût et à haut risque. Les solutions qui ne peuvent pas être améliorées relativement à un objectif sans dégrader les autres objectifs sont appelées les solutions Pareto optimales ; leur ensemble forme le front de Pareto. L'optimisation multi-objectif est largement appliquée dans de nombreux domaines de la science, y compris en économie, en finances et

en ingénierie.

0.1.3 La prise de décision séquentielle multi-objectif

Cette thèse est au carrefour de l'apprentissage par renforcement (RL) et l'optimisation multi-objectif (MOO). L'apprentissage par renforcement (RL) [Sutton and Barto, 1998; Szepesvári, 2010] est un domaine mature où de nombreux algorithmes avec des garanties d'optimalité ont été proposés au prix d'un passage à l'échelle quelque peu limité. Il traite des problèmes SDM dans le cadre de processus de décision de Markov (MDP). La recherche Monte-Carlo arborescente (MCTS), ancrée sur le cadre de bandit manchot, ou bandit à bras multiples (MAB) [Robbins, 1985], résout le problème du passage à l'échelle des algorithmes RL standard, avec d'excellents résultats dans nombreux problèmes de SDM de taille moyenne, comme des jeux [Ciancarini and Favini, 2009] et la planification [Nakhost and Müller, 2009]. Il procède par la construction itérative de l'arbre formalisant la séquence des décisions. Son efficacité algorithmique est notamment reconnue par son application au jeu de Go ; le programme MoGo a été salué comme une avancée fondamentale dans le domaine du jeu de Go par ordinateur [Gelly and Silver, 2007].

Motivée par le fait que de nombreuses applications dans le monde réel sont naturellement formulées dans le cadre de l'optimisation multi-objectif (MOO), cette thèse étudie le problème de la prise de décision séquentielle multi-objectif (MOSDM) où la récompense associée à un état donné dans le MDP est d-dimensionelle au lieu de scalaire. L'apprentissage par renforcement multi-objectif (MORL) a été appliqué aux tâches MOSDM telles que le contrôle du niveau d'eau du lac [Castelletti et al., 2002], l'équilibre entre la consommation d'énergie dans les serveurs web [Tesauro et al., 2007], planification de grille [Yu et al., 2008] et job-shop planification [Adibi et al., 2010].

0.2 Contributions Principales

Le présent travail concerne la prise de décision multi-objectif dans le cadre de MCTS. Il releve le défi de définir un règle de sélection de nœud lorsque les récompenses cumulées sont d-dimensionnelles, en s'appuyant sur des indicateurs bien étudiés de la littérature MOO. Les principales contributions sont les suivantes.

0.2.1 Algorithme MOMCTS

L'algorithme de la Recherche Monte-Carlo Arborescente Multi-Objectif (MOMCTS) a été proposé dans ce travail, dans lequel l'exploration de l'arbre MCTS a été modifié pour tenir compte de l'ordre partiel entre les nœuds dans l'espace d'objectif multidimensionnel, et le fait que le résultat souhaité est un ensemble de solutions Pareto-optimales (par opposition à une solution optimale unique).

Dans chaque nœud l'arbre de recherche de MOMCTS, une récompense vectorielle $\hat{\mathbf{r}}_{s,a} = (r_{s,a;1}, r_{s,a;2}, \cdots, r_{s,a;d})$ représentant la récompense moyenne dans chaque objectif est maintenue, ainsi que le nombre $n_{s,a}$ de visites sur le nœud. Chaque arbre dans MOMCTS est construit en suivant les trois mêmes phases que MCTS – la phase de sélection, la phase de construction de l'arbre et la phase aléatoire. Afin de s'adapter à la configuration MOO, les modifications apportées dans les trois phases sont présentées dans les sections suivantes.

La phase de sélection

La sélection de nœud MOMCTS dépend d'un score scalaire, qui définit un ordre total entre les nœuds avec des récompenses multi-dimensionnelles. Dans ce travail, nous proposons deux scores dans la phase de sélection de MOMCTS – l'indicateur de hypervolume et la récompense de dominance Pareto. Les deux scores appartiennent à la catégorie des fonctions de scalarisation fondées sur la population (section 4.2.3). Ils s'appuient sur l'archive P, qui maintient les récompenses vectorielles recueillies pendant le processus de recherche de MOMCTS.

La phase de construction de l'arbre

Dans la phase de construction de l'arbre, les heuristics d'élargissement progressif (Progressif Widening, PW) et d'estimation rapide de la valeur d'action (RAVE) qui sont optionellement utilisés dans MCTS (section 2.6.2) sont régulièrement intégrées dans MOMCTS. PW limite le nombre d'actions admissibles d'un nœud à une valeur entière $\lfloor n_{s,a}^{1/b} \rfloor$, avec b généralement fixé à 2 ou 4. La sélection de l'action dans la phase de construction de l'arbre repose sur l'heuristique RAVE.

La phase aléatoire

La phase aléatoire est réalisée de la même manière que dans MCTS, sauf que à la fin, une récompense vectorielle $\mathbf{R}$ est retournée. L'autre modification est que la fonction de

scalarisation basée sur la population qui maintient l'archive P des récompenses vectorielles reçues durant la recherche de MOMCTS[1] Sans perte de généralité, les points dominés sont supprimés de l'archive P.

MOMCTS

Par rapport à MCTS, le modification principlale apportée dans MOMCTS concerne l'étape de sélection de nœud. Le défi est d'étendre le critère mono-objectif de sélection de nœud au contexte multi-objectif. Comme indiqué, le noyau de la MOO est de récupérer l'ordre total entre les points de l'espace d'objectif multi-dimensionnel. La façon la plus simple de traiter avec l'optimisation multi-objectif est de revenir à l'optimisation mono-objectif, grâce à l'utilisation de la fonction de scalarisation. MOMCTS est caractérisé par la scalarisation des récompenses vectorielles basée sur la population des solutions précédentes, l'archive P. Contrairement à MCTS, qui estime la valeur de nœuds selon la distribution de récompenses fixe sur un seul objectif, MOMCTS estime la valeur de nœuds avec des récompenses à plusieurs dimensions en fonction de leur contribution à l'archive P. Notons que cette archive évolue au cours du processus, définissant un objectif non-stationnaire au long du processus de recherche.

Grâce à l'utilisation de la fonction de scalarisation basée sur la population, MOMCTS traite un problème d'optimisation mono-objectif dans chaque parcours d'arbre, dans lequel la qualité de l'ensemble des solution sauvegardées dans l'archive P est améliorée par la recherche répétitive de solutions simples. Plusieurs parcours d'arbres fournissent un ensemble de solutions optimales au sens de dominance Pareto dans MOMCTS.

L'algorithme MOMCTS est résumé par l'algorithme 0.1. Les hyper-paramètres communs à tous les algorithmes MOMCTS comprennent le budget de calcul N, le paramètre B utilisé dans l'heuristique d'élargissement progressive PW, et le modèle génératif $\mathcal{M}_D$ du problème MOSDM considéré. La valeur du nœud (s, a) noté par $g_x(s, a)$ est une fonction de scalarization basée sur la population, où x identifie le choix de la méthode de scalarisation.

Dans MOMCTS, l'estimation rapide de la valeur d'action (RAVE) prend une forme vectorielle ($\mathbf{RAVE}(a) \in \mathbb{R}^d, a \in \mathcal{A}$). Une fonction de scalarisation est donc nécessaire pour définir un ordre total entre les actions en se fondant sur l'estimation RAVE. Dans MOMCTS, la valeur scalarisée des vecteurs RAVE $g_{x;rave}(a), a \in \mathcal{A}$ se fonde sur la même fonction de scalarisation $g_x(s, a)$. La description des fonctions $g_x(s, a)$ et $g_{x;rave}(a)$ est donnée dans les sections 5.2 et 5.3.

Une propriété importante de MCTS est la propriété de consistance définie comme la capacité de l'algorithme de converger vers la politique optimale lorsque le nombre de parcours d'arbres N tend vers l'infini [Berthier et al., 2010]. La propiété de consistance est vérifiée dans le cas stationnaire, i.e. lorsque la distribution de la fonction récompense est

[1]Lorsque le nombre d'objectifs est faible ($d \leq 3$), les ressources de calcul et de mémoire nécessaires pour maintenir l'archive P sont limitées. Certaines heuristiques supplémentaires doivent être conçues pour préserver le passage à l'échelle de l'approche basée sur la population scalarisation dans le cadre de problèmes MOO faisant intervenir de nombreux objectifs (many objective optimization, MaOO). L'extension de MOMCTS au cas MaOO est une perspective de recherche future.

Algorithm 0.1: Algorithme MOMCTS

MOMCTS
Entrée: Nombre N de simulations
Sortie: Arbre de recherche $\mathcal{T}$
Initializer $\mathcal{T} \leftarrow$ racine (état initial), $P \leftarrow \{\}$
for $t = 1$ **to** N **do**
  Simulation($\mathcal{T}, P,$ root node)
end for
retourner $\mathcal{T}$

Simulation
Entrée: Arbre de recherche $\mathcal{T}$, archive P, nœud s
Sortie: récompense vectorielle $\mathbf{r}_u$
if s n'est pas une feuille, et $\neg(\lfloor (n_s+1)^{1/b} \rfloor > \lfloor (n_s)^{1/b} \rfloor)$ // (test PW non déclenché) **then**
  Selectionner $a^* = \arg\max\{g_x(s,a), (s,a) \in \mathcal{T}\}$
  $\mathbf{r}_u \leftarrow$ Simulation($\mathcal{T}, P, (s, a^*)$)
else
  $\mathcal{A}_s = \{$ actions disponibles non-visitées sous état $s\}$
  Selectionner $a^* = \arg\max\{g_{x;rave}(a), a \in \mathcal{A}_s\}$
  Ajouter (s, a^*) comme fils de s
  $\mathbf{r}_u \leftarrow$ SimulationAleatoire($P, (s, a^*)$)
end if
Mettre à jour n_s, n_{s,a^*}, $\hat{\mathbf{r}}_{s,a}$ et $\mathbf{RAVE}(a^*)$
retourner $\mathbf{r}_u$

SimulationAleatoire
Entrée: archive P, état u
Sortie: récompense vectorielle $\mathbf{r}_u$
$\mathcal{A}_{rnd} \leftarrow \{\}$ //sauvegarder l'ensemble des actions visitées durant la phase aléatoire
while u n'est pas l'état final **do**
  Selectionner uniformement une action disponible a pour u
  $\mathcal{A}_{rnd} \leftarrow \mathcal{A}_{rnd} \cup \{a\}$
  $u \leftarrow (u, a)$
end while
$\mathbf{r}_u \leftarrow \mathcal{M}_d(u)$ //obtenir la récompense vectorielle de la simulation
if $\mathbf{r}_u$ n'est pas dominé pas les points dans P **then**
  Eliminer tous les points dominés par r_u dans P
  $P \leftarrow P \cup \{\mathbf{r}_u\}$
end if
Mettre à jour $\mathbf{RAVE}(a)$ pour $a \in \mathcal{A}_{rnd}$
retourner $\mathbf{r}_u$

fixe au cours du temps. Dans le cas de MOMCTS, cependant, la fonction de scalarisation basée sur la population dépend de l'archive de P, et donc elle est non-stationnaire. L'étude de la consistance de l'approche proposée est une perspective de recherche future.

0.2.2 Indicateurs de qualité de solution multi-objectif

Les approches existantes en MORL [Gábor et al., 1998; Castelletti et al., 2002; Mannor and Shimkin, 2004; Natarajan and Tadepalli, 2005; Tesauro et al., 2007] sont pour la plupart basées sur la scalarisation linéaire de récompenses multidimensionnelles, avec la limitation qu'elle ne permet pas de découvrir des solutions sur les parties non-convexes du front de Pareto. Ces approches n'utilisent pas les indicateurs de qualité qui ont été définis et utilisés dans le domaine des Algorithmes Evolutionaires Multi-Objectif (MOEA) [Zitzler et al., 2003]. Ce travail établit un pont entre les deux domaines de MORL et MOEA, en introduisant deux de ces indicateurs d'évaluation des performances des politiques dans l'algorithme MOMCTS.

Spécifiquement, l'indicateur de hypervolume [Zitzler and Thiele, 1998] a été utilisé pour définir la performance scalaire d'un nœud. Comme montré par [Fleischer, 2003], l'indicateur de hypervolume est maximisée si et seulement si les points dans P^* appartiennent au front Pareto du problème MOO considéré. Auger et al. [2009] montrent que, pour $d = 2$, pour un certain nombre K de points, l'indicateur hypervolume projecte un problème d'optimisation multi-objectif défini dans $\mathbb{R}^d$, sur un problème d'optimisation mono-objectif dans $\mathbb{R}^{d\times K}$, dans le sens où il existe au moins un ensemble de K points dans $\mathbb{R}^d$ qui maximise l'indicateur hypervolume. Le mérite de cette approche est d'aller au-delà de la scalarisation linéaire standard. L'indicateur d'hyper-volume souffre toutefois de deux limitations. D'une part, les coûts de calcul d'indicateur de hypervolume augmentent de façon exponentielle avec le nombre d'objectifs. Deuxièmement, l'indicateur de hypervolume n'est pas invariant par la transformation monotone des objectifs. La propriété d'invariance (satisfaite par exemple par les algorithmes d'optimisation à base de comparaison) donne des garanties de robustesse extrêmement importantes pour les problèmes d'optimisation mal conditionnés [Hansen, 2006].

Par conséquent, un autre indicateur a été considéré : la récompense de dominance Pareto. Cette récompense peut être considé ré comme un compteur du nombre de découvertes de solutions non dominées, qui est cumulé de manière actualisée. Par rapport à la première approche − appelée MOMCTS-hv dans le reste de cette thèse, la deuxième approche − appelée MOMCTS-dom − a une complexité linéaire de calcul par rapport au nombre d'objectifs, et est invariante par rapport à la transformation monotone des objectifs. Le prix à payer pour l'amélioration de l'évolutivité de MOMCTS-dom est que la récompense de la dominance peut moins favoriser la diversité de l'archive Pareto, qui est une mesure essentielle de la qualité de l'ensemble de solutions non-dominés : un point non-dominé a la même récompense de dominance Pareto alors que l'indicateur de hypervolume favorise les points non-dominés situés dans les régions peu peuplées de l'archive Pareto.

0.2.3 Validation expérimentale

Les deux algorithms MOMCTS-hv et MOMCTS-dom ont été validés expéprimentalement sur quatre problèmes : Deep Sea Treasure (DST) [Vamplew et al., 2010], Resources Gathering (RG) [Barrett and Narayanan, 2008], Grid Scheduling [Yu et al., 2008] et Physical Travelling Salesman Problem (PTSP) [Powley et al., 2012]. Les deux premiérs problèmes artificiels sont conçus pour comparer les approches MOMCTS à l'état de l'art en MORL (les méthodes basées sur la scalarisation liné aire). Les deux dernièrs problèmes plus applicatifs sont utilisé pour tester le passage à l'échelle de MOMCTS. Les proprietés des problèmes considérés sont résumées par la Table 1.

Table 1: Problèmes de la prise de décision séquentielle multi-objectif

Problème	Forme du front Pareto	Fonction de transition déterministe or non-déterministe	Nombre d'objectifs	Decision en temps réel
Deep Sea Treasure	Non-convexe	Deterministe et Non-déterministe	2	Non
Resource Gathering	Convexe	Non-déterministe	3	Non
Grid Scheduling	Inconnu	Deterministe	2	Non
Physical Travelling Salesman	Inconnu	Deterministe	7	Oui

Les résultats expérimentaux sur le problème de Deep Sea Treasure confirment un mérite principal des approches proposées, leur capcité de découvrir des politiques se trouvant dans les régions non-convexes de front dePareto. À notre connaissance, cette fonctionnalité est unique dans la littérature MORL. Les expériences sur le probléme de Resource Gathering montrent que MOMCTS-dom bénéficie d'un meilleur passage à l'échelle que MOMCTS-hv en raison du coût de calcul de test de dominance Pareto qui est linéaire par rapport au nombre d'objectifs. Cette robustesse de MOMCTS-dom est en outre confirmée par les Physical Travelling Salesman Problem expériences, dont 7 objectifs sont optimisés de manière on-line.

En contrepartie, les approches MOMCTS souffrent de deux faiblesses principales. Tout d'abord, comme indiqué sur le Grid Scheduling et Physical Travelling Salesman Problem, une certaine connaissance préalable est nécessaire pour appliquer une exploration de MOMCTS avec efficacité. Deuxièmement, comme témoigné par le problème de Resource Gathering, les approches présentées découvrent peu de politiques "à risque" qui se trouvent dans une région peu prometteuse (la découverte d'optima de type "chapeau mexicain").

En conclusion, ce travail peut être considéré comme une preuve de concept de l'application du cadre MOMCTS pour les problème MOO. Les résultats obtenus peuvent être considérés comme prometteurs : les performances sont décentes comparativement à l'état de l'art, en dépit du fait qu'il s'agit d'approches beaucoup moins matures que les approches RL standard.

0.3 Les perspectives futures

Ce travail ouvre plusieurs perspectives de recherche, de type à la fois théorique et applicatif.

La perspective théorique principale concerne l'analyse des propriétés du mécanisme de mise à jour de la récompense cumulable dans le contexte général de l'optimisation (mono-objectif) dynamique. En outre, l'analyse de la consistance des critères de sélection de nœud actuels (y compris l'indicateur d'hypervolume et la récompense de dominance Pareto) permettra de définir des lignes directrices pour la conception de nouvelles récompenses scalarisées dans le cadre MOMCTS.

Du côté applicatif, d'une part, la fonction de préférence de scalarization linéaire utilisé dans les expérience de Physical Travelling Salesman Problem peut être étendue à un contexte plus général (par exemple non-linéaire), ce qui peut permettre à l'utilisateur d'exprimer ses préférences d'une façon plus naturelle et interactive.

Un perspective algorithmique concerne l'ajustement du mécanisme de mise à jour cumulatif actualisé de la récompense de dominance Pareto (Equation (5.7)). Vu que la découverte de solutions non-dominées est de plus en plus rare au cours du temps, l'ajustement du paramètre d'actualisation δ devrait être dynamique pour compenser cet effet de rareté. Une approche serait de considérer la découverte de nouvelles solutions non-dominées dans le cadre de la théorie de la valeur extrême [De Haan and Ferreira, 2007], et d'ajuster δ en conséquence.

Acknowledgements

First and foremost I want to thank my advisors, Michèle and Marc. I appreaciate all their contributions of time, ideas, and funding to make my Ph.D. experience productive and stimulating. Their encouragement, supervision and support enabled me to grow up as a Ph.D. for independently carrying out research. During my Ph.D. pursuit, they taught me how to do research, gave me suggestions when I met problems. I benefited a lot from their profound knowledge and rigorous attitude toward scientific research.

I deeply thank the reviewers of my dissertation, Prof. Jin-Kao Hao and Prof. Philippe Preux. Their comments and suggestions are very constructive for improving this dissertation.

I would like to thank all my friends from the TAO team and especially: Adrien, Dawei, Guohua, Ilya, Jean-Baptiste, Jialin, Lovro, Nicolas, Olivier, Ouassim and Riad. I am especially grateful to Ilya and Jean-Baptiste for sharing their experience and for many fruitful discussions we had.

I also thank all my closest friends and classmates: Heng, Minghao, Nicolas, Peiluo, Qingshan, Qiong and Zhengnan. The happy time we spent together stays forever in my memory.

I give my thanks to my parents who have always unconditionally supported me and cared me in all my pursuits. Lastly, I thank my wife Jingyi, for her love, support, encouragement, and companionship. You are my source of confidence and inspiration. Without you, it wouldn't have been possible for me to reach the step today.

To Jingyi

Contents

0.1 Contexte / Motivation i
0.1.1 Décision séquentielle i
0.1.2 Optimisation multi-objectif i
0.1.3 La prise de décision séquentielle multi-objectif ii
0.2 Contributions Principales iii
0.2.1 Algorithme MOMCTS iii
La phase de sélection iii
La phase de construction de l'arbre iii
La phase aléatoire iii
MOMCTS iv
0.2.2 Indicateurs de qualité de solution multi-objectif vi
0.2.3 Validation expérimentale vii
0.3 Les perspectives futures viii

Part I General Introduction

1 Introduction **3**
1.1 Context/Motivation 3
1.1.1 Sequential decision making 3
1.1.2 Multi-objective Optimization 4
1.1.3 Multi-objective sequential decision making 6
1.2 Main Contributions 7
1.3 Thesis Outline 8

Part II State of the art

2 Sequential Decision Making **11**
2.1 Modeling SDM problems 11
2.1.1 Markov Decision Process 12
2.1.2 Generative model 13
2.2 Goal of SDM 14
2.3 Reinforcement learning 14
2.3.1 RL framework 15

2.3.2 Value function based methods . . . 18
2.3.2.1 Basic dynamic programming algorithms . . . 19
2.3.2.2 Approximated algorithms . . . 22
2.4 Direct policy search . . . 24
2.5 Exploration vs. exploitation dilemma . . . 24
2.5.1 MAB settings . . . 25
2.5.2 Optimality criteria . . . 26
2.5.3 MAB Algorithms . . . 26
2.6 Monte-Carlo Tree Search . . . 28
2.6.1 MCTS algorithm . . . 28
2.6.2 MCTS extensions . . . 29

3 Multi-Objective Optimization **35**
3.1 MOO formal background . . . 35
3.1.1 Problem statement . . . 35
3.1.2 MOO optimality . . . 35
3.1.3 An MOO example . . . 37
3.1.4 MOO critical issues . . . 37
3.2 Classification of MOO approaches . . . 38
3.2.1 Weighted sum method . . . 39
3.2.2 ϵ-constraint method . . . 41
3.2.3 Goal programming technique . . . 43
3.2.4 Discussion . . . 43
3.3 Multi-Objective Evolutionary Algorithms . . . 43
3.3.1 Evolutionary algorithms . . . 44
3.3.2 Early MOEAs . . . 45
3.3.3 Quality indicators . . . 45
3.3.4 Unary quality indicators . . . 47
3.3.5 Modern MOEAs . . . 48
3.3.6 NSGA-II algorithm . . . 49
3.3.7 SMS-EMOA algorithm . . . 52
3.4 Discussion . . . 52

4 Multi-Objective Reinforcement Learning **55**
4.1 MORL background . . . 55
4.1.1 MOMDP . . . 55
4.1.2 Multi-objective generative model . . . 56
4.1.3 MOMDP difficulties . . . 56
4.2 Scalarization functions . . . 56
4.2.1 Linear scalarization functions . . . 57
4.2.2 Non-linear scalarization functions . . . 57
4.2.3 Population-based scalarization functions . . . 58
4.3 Single-policy MORLs . . . 59
4.3.1 Linear scalarization based single-policy algorithms . . . 59

4.3.2 Non-linear scalarization based single-policy algorithms 59
4.4 Multiple-policy MORLs 60
4.4.1 Linear scalarization based multiple-policy algorithms 60
4.4.2 Non-linear scalarization based multiple-policy algorithms 63
4.4.3 Population-based multiple-policy algorithms 63
4.5 MORL applications 64
4.5.1 Application scenarios 64
4.5.2 Real-world applications 66

Part III Contributions

5 Multi-Objective Monte-Carlo Tree Search 71
5.1 Overview of MOMCTS 71
5.1.1 Selection phase 71
5.1.2 Tree building phase 71
5.1.3 Random phase 72
5.1.4 MOMCTS framework 72
5.1.5 Discussion 72
5.2 MOMCTS based on hypervolume indicator 74
5.2.1 Hypervolume indicator based value estimation 74
5.2.2 MOMCTS-hv algorithm 75
5.2.3 Discussion on MOMCTS-hv 76
5.3 MOMCTS based on Pareto dominance reward 76
5.3.1 Cumulative discounted dominance reward based value estimation . . 77
5.3.2 MOMCTS-dom algorithm 78
5.4 Proof of concept 78
5.4.1 Example problem 78
5.4.2 Hypervolume indicator based criterion analysis 79
5.4.3 CDD reward analysis 81
5.4.4 Discussion 84

6 Experimental Analysis 85
6.1 Goals of experiments 85
6.2 Deep Sea Treasure problem 86
6.2.1 Problem statement 86
6.2.2 Experimental setting 87
6.2.3 Results 88
6.3 Resource Gathering problem 92
6.3.1 Problem statement 93
6.3.2 Experimental setting 93

6.3.3 Results . . . 96
6.4 Grid Scheduling problem . . . 99
6.4.1 Problem statement . . . 100
6.4.2 Experimental setting . . . 101
6.4.3 Results . . . 102
6.5 Physical Travelling Salesman Problem . . . 105
6.5.1 Problem statement . . . 105
6.5.2 Problem analysis . . . 106
6.5.2.1 Problem decomposition . . . 108
6.5.2.2 Macro actions . . . 109
6.5.2.3 Varying preference modes . . . 110
6.5.3 Baseline algorithms . . . 111
6.5.4 Experimental setting . . . 111
6.5.5 Results . . . 112
6.6 Partial conclusion . . . 116

Part IV Conclusion and Perspectives

7 Conclusions **121**
7.1 Summary of contributions . . . 121
7.2 Future directions . . . 122

List of Figures

1.1 An example schedule of a project presented in the form of Gantt chart. Only task dependencies and time allocations are presented in this schedule. Complete project schedule contains other resource allocation plans such as labor distribution and monetary budget for tasks. 4

1.2 Illustration of a navigation problem. The robot is required to go through the maze with walls (marked by yellow color) to reach the designated target way point. This figure shows a feasible solution (the path marked by consecutive black circles) for this navigation task. The solution involves the steering decisions in each time step, required to ultimately reach the goal. 5

1.3 Multi-objective descriptions in economics. Left: Three indifference curves showing three levels of satisfactions that different combinations of goods X and Y can bring to the customer. Right: An example production-possibility frontier (PPF) with illustrative points marked, among which Point D is said to Pareto dominate point A (more in Chapter 3) and Point X is outside the production possibility. Due to the law of diminishing marginal effect, the slope of the PPF curve decreases with the quantity of butter production. . 6

2.1 The interaction between the agent and the environment 12

2.2 Example of a Markov decision process. States are represented in blue circles and actions are represented by arrows. The transition probability between states are marked on the arrows, together with the associated rewards. . . . 13

2.3 An intuitive illustration from [Thiéry, 2010] shows how value iteration and policy iteration algorithms search for the optimal value function. According to Bertsekas and Tsitsiklis [1995], the value space can be separated into several polyhedrons, each corresponding to a value range where some policy (π_a, π_b or π^*) is greedy. We suppose that the state space $\mathcal{S}$ contains only two states s_1 and s_2, and the value space is therefore a plane. In policy iteration, each the policy evaluation step searches for the value function $V^{\pi_{t+1}}$. Through several alternative policy evaluation and policy improvement steps, this algorithm reaches the optimal value. In value iteration, the search is realized by a set of small steps which approach the optimal value function progressively. 21

2.4 The search tree and three phases of the MCTS algorithm. 28

3.1 Three non-dominated sets partitioned according to their Pareto ranks. . . . 37

3.2 A set of solutions (Left) and their correspondent objective values (Right) for the bi-objective example problem (Eq. 3.2), among which the optimal Pareto front is marked by red circles. The red points belong to the optimal Pareto front. The set of green points is non-dominated in the sense of Definition 7, although these solutions are dominated by the red ones. 38

3.3 Two sets of non-dominated solutions may be incomparable in the multi-objective space. 39

3.4 Left: the preference based approach of MOO. Note that the number of hyper-parameters ($|\mathbf{w}| = l$) in the user preference function is not necessarily the same as the number of objectives d. Right: the ideal approach of MOO. This illustration is adapted from [Tušar, 2007]. 40

3.5 Illustration of user preference function and the weighted sum method. (a) The isolines of user preference functions are depicted as the dashed contours. Points on the same dashed curve bring the same level of satisfaction to the user. (b) The user preference function is approximated by a weighted sum ($w_1 = w_2 = 0.5$) of objective values, the isolines of which are represented as dashed lines. (c) The points that maximize the weighted sum utility function under different weight settings are marked by red circles. The point lying in the non convex region of the Pareto front (marked by the black circle) does not maximize any linear combination of objective functions. 42

3.6 The basic evolutionary algorithm involves 4 steps: initialization, selection, variation and evaluation. Starting from a randomly or heuristically *initialized* population, all individuals in the population participate in a *selection* process which chooses better evaluated solutions to reproduce themselves. The chosen individuals (parents) go through the *variation* process (mutation and crossover) and generate off-springs. The off-spring population is then *evaluated*. According to the order defined on the evaluation results, the best individuals out of the off-spring (and optional parent) population are *selected* deterministically or stochastically and become the new generation population. The entire evolution process iterates until the stopping criterion is met. 44

3.7 With respect to the optimal Pareto front (red stars), the set B (blue squares) corresponding to a smaller generational distance is dominated by set A (green circles) with a greater generational distance. 48

3.8 The hypervolume indicator of the point set w.r.t. reference point z in the lower-left corner is the surface of the *shaded region*. 49

3.9 Calculation of the crowding distance. 50

4.1 The convex hull of points in the objective space. Each point in (a) represents the bi-objective value of a given policy and each line in (b) shows the dual representation of these points in the space of linearly scalarized policy values with the x-axis representing the weight w_1 for objective 1 (then $w_2 = 1-w_1$), and the y-axis representing the scalarized value of the policies. The convex hull is shown as circles in (a), and solid lines in (b). The Pareto front consists of all circles and the blue square in (a), among which the non-dominated points are marked by the red shadow, and points belonging to the convex hull are marked by the blue shadow. Notice that the blue square, representing a non-dominated point, does not belong to the convex hull because it lies in the non-convex region of the Pareto front. The gray stars in (a) and dashed lines in (b) are dominated points. 62

4.2 The three application scenarios for MORL. (a) Known preference scenario; (b) Varying preference scenario; (c) Decision support scenario (adapted from [Roijers et al., 2013]). 65

5.1 Left: For a vectorial reward $\bar{\mathbf{r}}_{s,a}$ that is not dominated w.r.t. the archive P, $V_{hv}(s,a)$ is its hypervolume indicator contribution to the solution set. Right: For a vectorial reward that is dominated by some point in the archive P, its value is measured by the opposite of its Euclidean distance to the approximated surface (dashed lines) of the Pareto front. 75

5.2 Reward distributions of the bi-objective MAB problem. The reward regions corresponding to action $1, 2$ and 3 are respectively marked by green, blue and purple shadows. 79

5.3 The evolution of hypervolume indicator rewards in action $1, 2, 3$ of representative runs under different EvE trade-off parameter settings. 80

5.4 The approximated Pareto surface found under the hypervolume indicator based action selection criterion with $c_r = c_{r'} = 0.1$. The evolution of the empirical Pareto front has a non-smooth impact on the hypervolume indicator base value estimation (e.g. $\bar{\mathbf{r}}_3^p$ jumps forward when the Pareto front moves). 81

5.5 The Pareto optimal solution set found under the CDD based action selection criterion gradually moves towards the true Pareto front. 82

5.6 The evolution of CDD rewards in action $1, 2, 3$. 83

6.1 The Deep Sea Treasure problem. Left: the DST state space with black cells as sea-floor, gray cells as terminal states, the treasure value is indicated in each cell. The initial position is the upper left cell. Right: the Pareto front in the time×treasure plane. 87

6.2 The hypervolume indicator performance of MOMCTS-hv, MOMCTS-dom and MOQ-learning versus training time in the deterministic DST problem. For the sake of a fair comparison with MOQ-learning, the training time refers to the number of action selections in MOMCTS approaches (each tree-walk in MOMCTS carries out on average 7 to 8 action selections in the DST problem). Top: The hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning-m=21. Bottom: The hypervolume indicator of MOQ-learning with $m = 3, 7, 21$. 89
6.3 Left: The vectorial rewards found by representative MOMCTS-hv, MOMCTS-dom and MOQ-learning-m = 21 runs. Right: The percentage of times out of 11 runs that each non-dominated vectorial reward was discovered by MOMCTS-hv, MOMCTS-dom and MOQ-learning-m = 21, during at least one test episode. 90
6.4 The hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning-m=21 versus training time in the stochastic environment with (a) $\eta = 0.01$ and (b) $\eta = 0.1$. 91
6.5 The Resource Gathering problem. The initial position of the agent is the mid-bottom case. Two resources (gold and gems) are located in fixed positions. Two enemy cases (marked by swords) send the agent back home with 10% probability. 92
6.6 The seven policies in the Resource Gathering problem that correspond to the non-dominated vectorial rewards. 94
6.7 The seven non-dominated vectorial rewards in the Resource Gathering problem identified by Vamplew et al. [2010]. 94
6.8 The Resource Gathering problem: Average hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning (with m = 6, 15 and 45) over 11 runs, versus number of time steps. The optimal hypervolume indicator 2.01×10^{-3} is indicated by the top line. 96
6.9 The vectorial rewards found by representative MOMCTS-hv, MOMCTS-dom and MOQ-learning with $m = 6, 15$ runs. Left: the points projected on the $Gems = 0$ plane. Right: the points projected on the $Gold = 0$ plane. The Pareto optimal points are marked by circles. 97
6.10 The percentage of of times out of 11 runs that each non-dominated vectorial reward was discovered by MOMCTS-hv,MOMCTS-dom and MOQ-learning with $m = 6, 15, 45$, during at least one test period. 97
6.11 The hypervolume indicator performance of MOMCTS-dom with δ varying in $\{0.9, 0.98, 0.99, 0.999\}$, versus training time steps in the Resource Gathering problem. 98
6.12 The Resource Gathering problem: average computational cost for one tree-walk for MOMCTS-hv and MOMCTS-dom over 11 independent runs. On average, each tree-walk in MOMCTS is ca. 35 training time steps. 99
6.13 Scheduling a job containing 7 interdependent tasks on a grid of 2 resources. Left: The dependency graph of tasks in the job. Right: The illustration of an execution plan. 100

6.14 The EBI_ClustalW2 workflow. 101
6.15 The generational distance (GD) and inverted generational distance (IGD) for $N = 100$, 1000 and 10000 of MOMCTS-hv, MOMCTS-dom, NSGA-II and SMS-EMOA on (a): EBI_ClustalW2; (b)(c)(d): artificial problems with number of tasks J and graph density q. Each performance point after 1000 and 10 000 evaluations are respectively marked by single and double circles. 103
6.16 Progression of the Pareto-optimal solutions found for $N = 100$, 1000 and 10000 for MOMCTS-hv, MOMCTS-dom, NSGA-II and SMS-EMOA on the EBI_ClustalW2 workflow. The reference Pareto front is indicated by circles. 104
6.17 The PTSP map. 105
6.18 Action space of PTSP. 106
6.19 Legends of elements in an PTSP map. 107
6.20 An example path which traverse all way-points in the PTSP map. 1530 actions are made to correct the direction and accelerate the ship in this path. 107
6.21 Paths with different visit orders corresponds to different length in PTSP extracted from [Powley et al., 2012]. Figures b and d respectively shows the trace of the ship by following orders defined by Figure a and c. Note that the time is not simply determined by the travel length due to the inertia effect. 108
6.22 Examples of the path followed by the MOMCTS controller. Left: The path created with $M = 10$, in which rotations are smooth. Right: The path created with $M = 30$, in which most turns are in 90° angle. 109
6.23 Three representative maps extracted from the 2013 CIG PTSP competition toolkit. All three maps are of the same size 512 pixel × 512 pixel. Blue points are way-points to be visited, and green points are fuel tanks. The difference between these maps lies in the obstacle setting. The simple map (a) does not contain any obstacle, while the medium map (b) contains more rugged walls and 4 obstacles in the middle. The difficult map (c) is a maze-like arena (black segments indicate walls) in which no straight path exist between any two way-points. 113
6.24 Sensitivity analysis: impact of macro-action length M on hypervolume indicator performance of MOMCTS, MCTS and MC in simple, medium and difficult maps. 114
6.25 The result vector $(R_{time}, R_{damage}, R_{fuel})$ distribution of MOMCTS, MCTS and MC in the three tested maps. 117

List of Tables

1 Problèmes de la prise de décision séquentielle multi-objectif vii

4.1 Summary of applications of MORL to real-world problems. 67

5.1 Action selection frequencies among 3000 time steps in the bi-objective MAB problem (averaged over 11 runs). 82

6.1 Multi-objective SDM problems 86
6.2 The DST problem: hypervolume indicator results of MOMCTS-hv, MOMCTS-dom and MOQ-learning with m ranging in 3,7 and 21 with different noise levels η, averaged over 11 independent runs after 300,000 time steps. The optimal hypervolume indicator is 10455. For each η, best results are indicated in bold font (significance value $p < 0.05$ according to the Student's t-test). 88
6.3 The optimal policies for the Resource Gathering problem. 95
6.4 The Resource Gathering problem: Average hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning (with $m = 6$, 15 and 45) over 11 runs. The optimal hypervolume indicator is 2.01×10^{-3}. Significantly better results are indicated in bold font (significance value $p < 0.05$ for the Student's t-test). 96
6.5 The best hypervolume indicator of the set of result vectors $\mathbf{R} = (R_{time}, R_{damage}, R_{fuel})$ obtained by MOMCTS, MCTS and MC over 11 runs (each run generates one single result vector $\mathbf{R}$) in the three test maps, with the reference point z set to $(5000, 5000, 5000)$. The best results are indicated in bold font. 112

List of Algorithms

0.1 Algorithme MOMCTS v

2.1 Value iteration 19
2.2 Policy iteration algorithm 20
2.3 Q-learning algorithm 23
2.4 SARSA algorithm 23
2.5 Multi-armed bandit framework 25
2.6 MCTS algorithm 30

3.1 Fast non-dominated sort algorithm 46
3.2 $(\mu + \lambda)$-NGSA-II algorithm 51
3.3 TournamentSelection function 51

4.1 TLQ-learning 60
4.2 Hypervolume-based Q-learning algorithm 64

5.1 MOMCTS framework 73

List of Acronyms

DP Dynamic Programming

EA Evolutionary Algorithm

EvE Exploration vs. Exploitation

MAB Multi-Armed Bandit

MCTS Monte-Carlo Tree Search

MDP Markov Decision Process

MOEA Multi-Objective Evolutionary Algorithm

MOMCTS Multi-Objective Monte-Carlo Tree Search

MOO Multi-Objective Optimization

MORL Multi-Objective Reinforcement Learning

RL Reinforcement Learning

SDM Sequential Decision Making

UCB Upper Confidence Bound

UCT Upper Confidence Bounds applied to Trees

Part I

General Introduction

Chapter 1

Introduction

This thesis is concerned with the multi-objective sequential decision making problem. Firstly, sequential decision making algorithms, more specifically the Monte-Carlo tree search (MCTS) [Kocsis and Szepesvári, 2006], have been thoroughly studied in this work. Secondly, studies in multi-objective optimization are explored and integrated into MCTS.

1.1 Context/Motivation

1.1.1 Sequential decision making

Decision making composes an important part of daily activities in our life. Decision making usually relies on a total order measure (such as reward function), indicating the quality of decisions to be optimized. The sequential decision making (SDM) problem is more complex in the sense that optimal decision sequences, a.k.a. policies, are usually not formed by selecting the best individual decision in each step, and the decisions composing the optimal sequence are interdependent. A typical example of this is that of games [Aliprantis and Chakrabarti, 2000], scheduling [Zhang and Dietterich, 1995] [1], or robotics [Mahadevan and Connell, 1992].

Sequential decision making is frequently used in strategic games, such as chess and go, in which players alternatively move or place their pieces on a game board. In chess, if digital scores are associated to different pieces in game, as was done in many chess-playing computer projects, including the famous IBM *Deep Blue* computer [Hsu, 2002], the piece moving decisions can be made automatically by choosing the move that maximizes a heuristic evaluation function, such as the sum of scores of all pieces on board in the end of the game. In the game of Go however, no such good heuristic evaluation function is available, which is primarily related to the fact that the influence of moving one piece anc only be seen after a long delay (and secondarily to the fact that the number of relevant moves is much higher in Go than in chess).

Let us note the average number of candidate decisions in each turn of the game by B, and the total number of turns in the game by T. Then there are B^T possible decision sequences in the game. Selecting the optimal decision sequence among an exponential number of candidate sequences w.r.t. the decision length compose the main difficulty of SDM in games.

Additionally, in some real-world problems, like project scheduling (Figure 1.1) and robotic navigation (Figure 1.2), the stochasticity in the envionment implies that the quality

[1] Scheduling can also be handled as a combinatorial optimization problem [Brucker and Brucker, 2007].

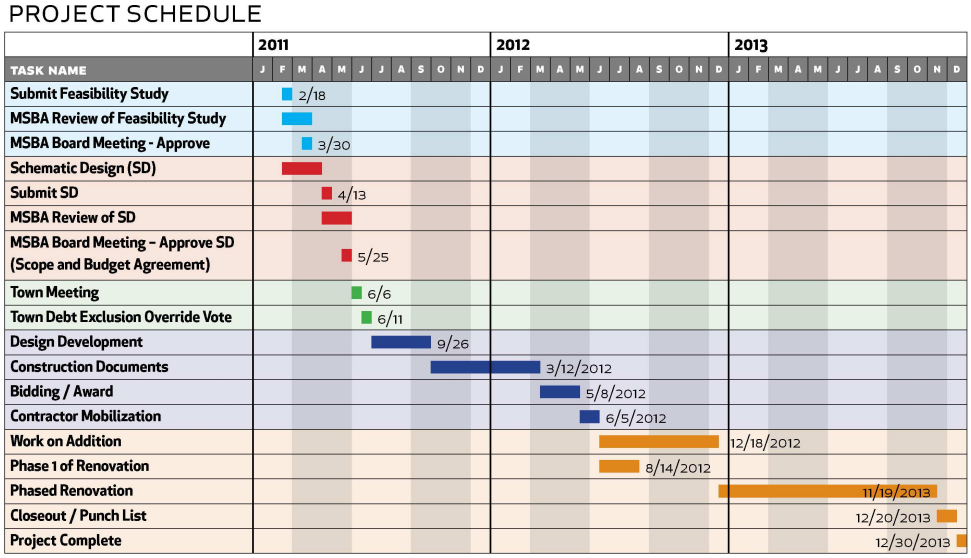


Figure 1.1: An example schedule of a project presented in the form of Gantt chart. Only task dependencies and time allocations are presented in this schedule. Complete project schedule contains other resource allocation plans such as labor distribution and monetary budget for tasks.

of a policy should be maximized in expectation, calling for extensive computation and memory resources to be estimated.

In all cited applications, it is clear that the optimal decision to sequential decision making problems (strategic thinking, in the context of games; avoiding traps in robotic navigation) can hardly be obtained by finding the locally optimal solution, due to the problem of delayed rewards. The purpose of reinforcement learning (Chapter 2) is to yield globally optimal decision sequences, which requires to identify the value of decisions in the long term perspectives.

1.1.2 Multi-objective Optimization

Independently, many real-world decision problems involve multiple objectives (e.g. the manufacturing process which simultaneously minimizes the cost and the risk), and these problems are referred to as Multi-Objective Optimization (MOO). For a non-trivial MOO problem, there does not exist a single solution that simultaneously optimizes each objective. The objective functions are conflicting, two solutions are not necessarily comparable, for instance, one production plan might be of high cost and low risk, while the other is of low cost and high risk. The set of solutions which can not be improved in one objective without deteriorating other objectives are called the Pareto optimal solutions (see Figure 1.3, more in Chapter 3). Multi-objective optimization is widely applied in many fields of science, including economics, finance and engineering.

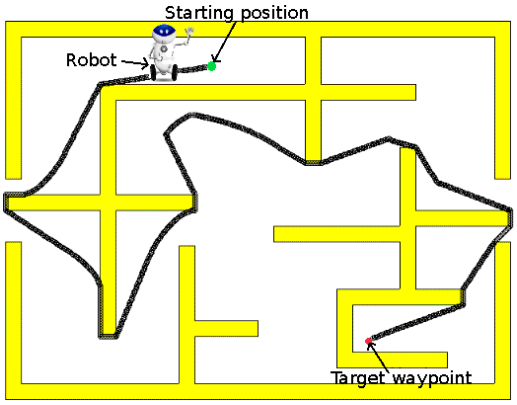


Figure 1.2: Illustration of a navigation problem. The robot is required to go through the maze with walls (marked by yellow color) to reach the designated target way point. This figure shows a feasible solution (the path marked by consecutive black circles) for this navigation task. The solution involves the steering decisions in each time step, required to ultimately reach the goal.

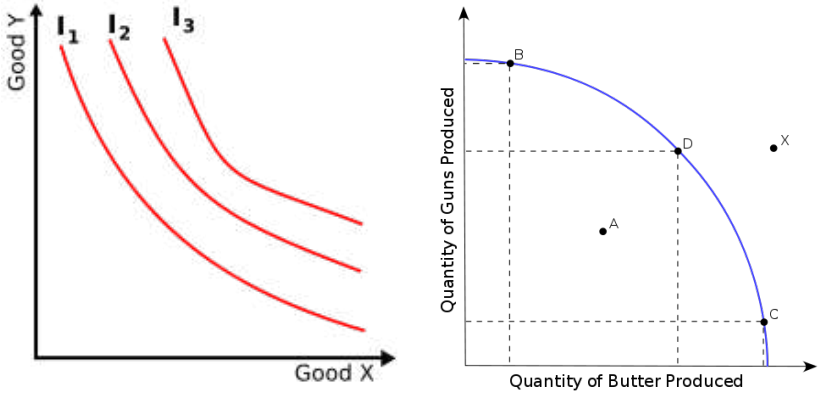
Economics Economics is the application field where multi-objective optimization was originated from. In microeconomics theory, people use the indifference curve to show different bundles of goods (objectives) between which a consumer is indifferent (Figure 1.3(a)). Each point on the indifference curve is considered as rendering the same level of utility (reward or satisfaction) for the consumer. Vilfredo Pareto, after whom a series of notions in MOO study today (such as *Pareto dominance, Pareto front*, more in Chapter 3) are named, was the first author to draw the indifference curves in his book *Manual of Political Economy*, 1906.

Economists also use the production-possibility frontier (PPF) (Figure 1.3(b)) to describe various combinations of amounts of two commodities that could be produced using the fixed amount of resource (land, labor, capital, time, etc.). Under a limited resource budget, maximizing the customer utility and extending the PPF both compose multi-objective optimization problems.

Finance In finance, a common problem is to choose a portfolio when there are two conflicting objectives — maximizing the expected value of portfolio returns, and minimizing the risk, measured by the standard deviation of portfolio returns. The bi-objective problem of maximizing the expected value (first moment) and minimizing the standard deviation (square root of the second moment) of portfolio return is extensively studied by [Meyer, 1987] under the name of a two-moment decision model.

Engineering MOO methods are frequently used in the optimal design and optimal control problems in engineering.

Firstly, a good design typically involves multiple objectives such as financial cost/investment, operating cost, profit, quality, efficiency, process safety, operation time,



(a) Indifference Curves (b) Production-possibility frontier

Figure 1.3: Multi-objective descriptions in economics. Left: Three indifference curves showing three levels of satisfactions that different combinations of goods X and Y can bring to the customer. Right: An example production-possibility frontier (PPF) with illustrative points marked, among which Point D is said to Pareto dominate point A (more in Chapter 3) and Point X is outside the production possibility. Due to the law of diminishing marginal effect, the slope of the PPF curve decreases with the quantity of butter production.

etc. Consequently, in practical applications, the performance of process and product design is often measured with respect to multiple criteria. These objectives typically are conflicting, and MOO techniques are therefore required.

Secondly, the controlling problem in engineering involves keeping the output of a system as close as possible to the target value. Target values of a system output usually involve several aspects, and are subject to constraints that prevent all objectives from being simultaneously perfectly met. For example, one might want to adjust a rocket's fuel usage and orientation so that it arrives both at a specified place and at a specified time. MOO methods are used to balance the distances between the system outputs and their desired values [Zhai et al., 2013].

1.1.3 Multi-objective sequential decision making

This thesis is at the crossroad of Reinforcement Learning (RL) and multi-objective optimization (MOO).

Reinforcement Learning (RL) [Sutton and Barto, 1998; Szepesvári, 2010], which will be presented in Chapter 2, is a mature field where many algorithms with optimality guarantees have been proposed at the expense of a somewhat limited scalability. It addresses the sequential decision making (SDM) problems in the Markov decision process (MDP) framework. Monte-Carlo tree search (MCTS), rooted on the multi-armed bandit (MAB) framework [Robbins, 1985], overcomes the scalability problem of standard RL for many medium size SDM problems, such as games [Ciancarini and Favini, 2009] and planning

[Nakhost and Müller, 2009]. It proceeds by iteratively building the tree formalizing the sequence of decisions (Chapter 2). Its algorithmic efficiency is in particular acknowledged through its application to the Computer Go player – Mogo, a breakthrough in the domain of computer go [Gelly and Silver, 2007].

Motivated by the fact that many real-world applications are naturally formulated in terms of multi-objective optimization (MOO), this thesis studies multi-objective sequential decision making (MOSDM) problem, where the reward associated to a given state in the MDP is d-dimensional instead of a single scalar value. multi-objective reinforcement learning (MORL) methods have been applied to MOSDM tasks such as lake water level control [Castelletti et al., 2002], balancing power consumption in web servers [Tesauro et al., 2007], grid scheduling [Yu et al., 2008] and job-shop scheduling [Adibi et al., 2010].

1.2 Main Contributions

The present work is concerned with multi-objective sequential decision making within the MCTS framework addressing the challenge of defining a node selection rule that can be extended to the multi-objective case, building upon indicators from the MOO literature. The main contributions are as follows:

1. A Multi-Objective Monte-Carlo Tree Search (MOMCTS) framework has been proposed in this work, in which the exploration of the MCTS tree has been modified to account for the partial order among the nodes in the multi-dimensional objective space, and the fact that the desired result is a set of Pareto-optimal solutions (as opposed to, a single optimal one).

2. Existing applications in MORL [Gábor et al., 1998; Castelletti et al., 2002; Mannor and Shimkin, 2004; Natarajan and Tadepalli, 2005; Tesauro et al., 2007] are mostly based on the linear-scalarization of multi-dimensional rewards, while ignoring the quality indicators which has been used in the MOEA setting [Zitzler et al., 2003]. This work fills the gap between MORL and MOEA by introducing two performance assessment indicators into the MOMCTS algorithm as the policy selection criteria.

   Specifically, the hypervolume indicator [Zitzler and Thiele, 1998] has been used to provide node rewards. The merit of this approach is to go beyond the standard linear-scalarization. This approach suffers from two limitations. On the one hand, the hypervolume indicator computation cost increases exponentially with the number of objectives. Secondly, the hypervolume indicator is not invariant under the monotonous transformation of the objectives. The invariance property (satisfied for instance by comparison-based optimization algorithms) gives robustness guarantees which are most important w.r.t. ill-conditioned optimization problems [Hansen, 2006].

   Therefore, another indicator has been considered – the Pareto dominance reward. Compared to the first approach − referred to as MOMCTS-hv in the remainder of this thesis, the latter approach − referred to as MOMCTS-dom − has linear computational complexity w.r.t. the number of objectives, and is invariant w.r.t. the

monotonous transformation of the objectives. The price to pay for the improved scalability of MOMCTS-dom is that the dominance reward might less favour the diversity of the Pareto archive, which is an essential measure of quality of non-dominated solution sets (Chapter 3) : any non-dominated point has the same dominance reward whereas the hypervolume indicator of non-dominated points in the sparsely populated regions of the Pareto archive is higher.

3. Both MOMCTS-hv and MOMCTS-dom algorithms have been experimentally validated on four problems : Deep Sea Treasure (DST) [Vamplew et al., 2010], Resource Gathering (RG) [Barrett and Narayanan, 2008], grid scheduling [Yu et al., 2008] and Physical Travelling Salesman Problem (PTSP) [Powley et al., 2012].

   The experimental results on DST confirm a main merit of the proposed approaches, their ability to discover policies lying in the non-convex regions of the Pareto front. To our knowledge, this feature is unique in the MORL literature. The experiments in the three-objective RG problem validate the scalability of MOMCTS-dom algorithm in higher dimensional optimization problems. Through comparative experiments of MOMCTS w.r.t. the state of the art in grid scheduling and PTSP 2013 international competition, the potential of MOMCTS framework in solving real-world problems has been shown.

1.3 Thesis Outline

The thesis manuscript is organized as follows.
Chapter 2 introduces the formal background of sequential decision making, focusing on the reinforcement learning and Monte-Carlo tree search.
Chapter 3 presents the basic notions, performance indicators and the popular techniques used in multi-objective optimization (MOO).
Chapter 4 introduces the state of the art and applications of multi-objective reinforcement learning (MORL).
Chapter 5 extends the MCTS framework to MOMCTS, presenting and discussing the hypervolume indicator and dominance reward as solution selection criteria.
Chapter 6 finally describes the validation results of MOMCTS on the Deep Sea Treasure (DST), Resource Gathering (RG), grid scheduling, and Physical Traveling Salesman Problem (PTSP).
Chapter 7 concludes the thesis and presents perspectives for further research.

Part II

State of the art

Chapter 2

Sequential Decision Making

In this chapter, we introduce the formal background of Sequential Decision Making (SDM). Then the state of the art in SDM – reinforcement learning (RL) is presented, with focus on the exploration vs. exploitation (EvE) dilemma. Finally, the Monte-Carlo tree search (MCTS) framework is presented.

2.1 Modeling SDM problems

The basic notions in SDM, after Sutton and Barto [1998] and Szepesvári [2010], can be illustrated in the navigation problem presented in section 1.1.1 (Figure 1.2), in which we search for a method that guides the robot towards the target position.

The different elements that compose an SDM problem are as follows:

- agent (the robot)
- environment (the arena)
- decision epochs (time steps)
- Markov decision process (MDP)
    - states (position, speed and sensor values)
    - actions (accelerating, reversing and turning)
    - transition function
    - rewards

The agent In SDM problems, the agent means the system which lives in the environment and makes decisions. Agents can also be software, equipments or any other decision-making entities. The agent takes the information about the environment and about itself as input, then it makes decisions and turns these decisions into actions which influence its own state and the environment (Figure 2.1). In the navigation example, the robot is the agent.

The environment The environment is anything external to the agent. In this work, we assume that the environment changes in response to the actions of the agent according to fixed procedures.

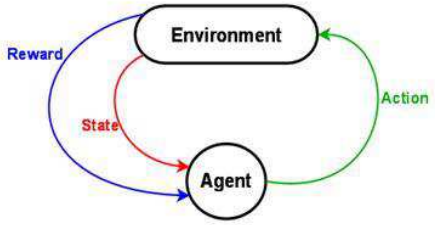


Figure 2.1: The interaction between the agent and the environment

Decision epochs The decision epochs are the times at which a decision should be made, i.e. at which the agent needs to choose an action. In this work, we only consider the case of discrete decision epochs (called time steps), referring the reader interested in the continuous time decision making to [Bertsekas et al., 1995]. Notice that discrete time steps do not necessarily mean equally spaced decision epochs. By convention, we note the value of a variable X at time t by X_t.

2.1.1 Markov Decision Process

SDM problems are usually modeled by Markov Decision Processes (MDP), containing four elements – a state space, an action space, a transition probability density function and a reward function. In MDPs, the state transitions possess the *Markov property*, which means that the transition probability towards the next state depends only on the current state and the action of the decision maker. It is independent of all previous states and actions, conditionally to the current state and action.

State space A state contains every information about the agent and the environment needed to make a decision. We generally refer to state variables as $s \in \mathcal{S}$, with $\mathcal{S}$ referring to the state space. The space $\mathcal{S}$ can be finite or continuous. In the navigation problem, $\mathcal{S}$ is a continuous space (if the robot position is a real valued vector) or a discrete space (in a grid world).

Action space The current action, selected by the agent depending on the current state, will change the state of the environment and the agent. The action space is the set of possible choices or agent decisions, like the motor instructions (acceleration and steering) in the navigation problem. We generally denote an action by $a \in \mathcal{A}$, $\mathcal{A}$ being the action space. Similar to the state space, the action space can be finite or continuous. In some SDM problems, only a subset of actions in $\mathcal{A}$ are possible to take under a certain state s, in this case, the set of *admissible* actions in state s is noted as $\mathcal{A}_s$.

Transition function The transition function f models the dynamics of how the state is modified under the action of the agent. There are basically two cases, the deterministic case and the stochastic case. In the deterministic case, state s_{t+1} is a function of the

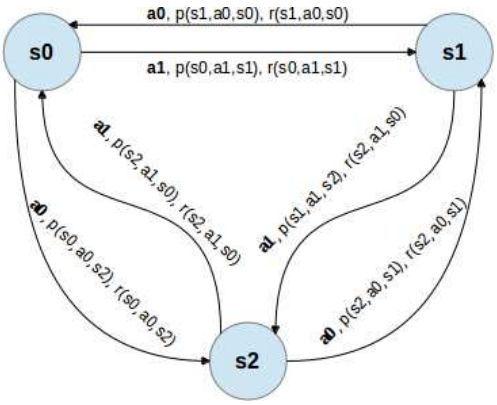


Figure 2.2: Example of a Markov decision process. States are represented in blue circles and actions are represented by arrows. The transition probability between states are marked on the arrows, together with the associated rewards.

agent's state s_t, and the action a_t at time t ($s_{t+1} = f(s_t, a_t)$). In the stochastic case, the transition function is defined through a probability function $p(s_t, a_t, s')$ yielding the probability of arriving in s' upon executing a_t under state s_t.

Rewards The last element in an MDP is the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, defining the instant reward the agent gets by selecting action a in state s. The bounded reward function is computed at each time step and is noted by $r_t = r(s_t, a_t)$.

Starting from an initial state s_0, the MDP specifies the possible interactions between the agent and the environment in discrete time. At a given time step, the agent in state $s \in \mathcal{S}$ chooses an action from the admissible action set $\mathcal{A}_s$. At the next time step, the agent will be in state s', drawn randomly by following the probability density $p(s, a, s')$, and receive reward $r(s, a)$. Figure 2.2 gives a graphical illustration of an MDP.

In an MDP, we define a policy as a function $\pi : \mathcal{S} \to \mathcal{A}$. Although stochastic policies have been considered in the literature [Kaelbling et al., 1996; Peters and Schaal, 2008], only deterministic policies will be considered in the following.

2.1.2 Generative model

The MDP can be given explicitly, or through a *generative model*. The generative model describes the transition function and reward function which might be available in some application domains, for instance, games. The generative model is a function $\mathcal{M} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathbb{R}$. Given the current state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}_s$, $\mathcal{M}$ returns the next state s' and the associated reward r which are deterministic or stochastic values.

Compared to the MDP, the generative model does not require the knowledge about the entire state space $\mathcal{S}$ and the action space $\mathcal{A}$, and is available in more practical applications than MDP.

2.2 Goal of SDM

Given the state s, the agent will cumulate rewards during its lifetime, where two cases are distinguished: finite lifetime, referred to as the time horizon T, and the infinite time horizon $T = \infty$. The influence of the finite and infinite time horizon case will be presented in the next section. Naturally, the SDM goal is to optimize the cumulative reward of the agent within the time horizon.

Several settings are distinguished among the SDM methods, depending on whether the environment is known or not. In the former case, SDM boils down to planning (or optimal control [Littman, 1996]). In the later case, a mainstream approach is reinforcement learning (RL) [Sutton and Barto, 1998; Szepesvári, 2010].

In planning, the transition model of the environment is known in advance. In order to produce credible plans, two conditions usually need to be met: a) the model of the world must be well defined and informative and b) the world must not deviate from the given model at any time point. In other words, the agent assumes that the world is stationary and can only be changed by its actions. Early work [Fikes and Nilsson, 1972] formulates planning as a search problem. Heuristic search algorithms, such as Hill-Climbing [Goldfeld et al., 1966] and A* [Hart et al., 1968] are used for solving it. In recent years, planning algorithms have undergone a rapid development of the search efficiency, leading to methods that are up to millions of times faster than A* algorithm [Delling et al., 2009].

Reinforcement Learning (RL) addresses the SDM problem through interactions with environment. Unlike the planning algorithms which require complete description of the environment states, actions, rewards, and transitions, RL can be used when a model of the environment is unknown . An ideal reinforcement learning agent does whatever it needs to find an optimal policy. This usually enforces the balance between learning the transition model of the considered problem, and finding an optimal policy on the basis of this model. What is the optimal balance between the two is yet an open problem. Therefore most reinforcement learning algorithms alternate between finding a good approximation of the problem model, and finding the optimal policy on the basis of the approximated model.

2.3 Reinforcement learning

Reinforcement learning (RL) has a strong ethological and cognitive science basis:

Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will – other things being equal – be more firmly connected with the situation, so that when it recurs, they will more likely to recur;

those which are accompanied or closely followed by discomfort to the animal will – other things being equal – have their connection with the situation weakened, so that when it recurs, they will less likely to recur;

the greater the satisfaction or discomfort, the greater the strengthening or weakening of the link. —Thorndike, 1911.

In standard RL, the agent must learn a generative model of the environment (the

transition and reward functions) and estimate the values of the states and actions.

In this section, we will introduce the notations used in RL, followed by the main RL approaches.

2.3.1 RL framework

Value function

The value of a policy π is a function $V^\pi : \mathcal{S} \to \mathbb{R}$ which associates to each state the expectation of cumulative rewards (also call *expected return*) that π gets when starting from this state. In finite time horizon case, which is also called episodic case, the value function is defined as the sum of the rewards in the next T time steps.

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{T} r(s_t, a_t) | s_0 = s, a_t = \pi(s_t)\right] \tag{2.1}$$

In infinite time horizon case,

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r(s_t, a_t) | s_0 = s, a_t = \pi(s_t)\right] \tag{2.2}$$

where $\gamma \in [0, 1[$ is the discount factor indicating that reward gathered at time step $t+1$ is less important that those gathered at time t, everything being equal. Note that γ also enforces the boundedness of V^π. In most RL algorithms, the time horizon T is infinite. If not specified, only infinite-time horizon value functions will be considered in the remainder of this chapter.

Action value function

The action value function $Q^\pi(s, a)$ (also called Q-value function) is defined in a similar way as the value function. It is the expectation of cumulative rewards after executing a in state s and thereafter following policy π.

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r(s_t, a_t) | s_0 = s, a_0 = a, a_t = \pi(s_t) \text{ for } t \geq 1\right] \tag{2.3}$$

Bellman equations and Bellman optimality

The Markov property in MDP is essential to RL because it allows establishing the Bellman equation which is the basis of many MDP solutions [Bellman, 1986].

Having the state and action values, the first Bellman equation determines the value of

any given policy through a fixed point equation:

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \cdot r(s_t, a_t | s_0 = s, a_t = \pi(s_t))\right] \\
&= r(s_0, a_0) + \mathbb{E}\left[\sum_{t=1}^{\infty}\gamma^t \cdot r(s_t, a_t) | s_0 = s, a_t = \pi(s_t)\right] \\
&= r(s, \pi(s)) + \mathbb{E}\left[\sum_{s' \in \mathcal{S}} p(s, \pi(s), s') \cdot \sum_{t=1}^{\infty}\gamma^t \cdot r(s_t, a_t) | s_0 = s, s_1 = s', a_t = \pi(s_t)\right] \\
&= r(s, \pi(s)) + \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') \cdot \mathbb{E}\left[\sum_{t=1}^{\infty}\gamma^t \cdot r(s_t, a_t) | s_1 = s', a_t = \pi(s_t)\right] \\
&\quad \text{(according to the Markov property, the state transition probability after } s_1 \text{ does not depend on } s_0) \\
&= r(s, \pi(s)) + \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') \cdot \gamma \cdot \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \cdot r(s_t, a_t) | s_0 = s', a_t = \pi(s_t)\right] \\
&= r(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') V^{\pi}(s')
\end{aligned} \tag{2.4}$$

where $r(s, \pi(s))$ denotes the instant reward obtained by executing action $\pi(s)$ under state s.

The above recursive equation reveals that the value of a state s depends on the immediate reward obtained by executing $\pi(s)$ and the value of the following states. It is the basis of many algorithms which search for the value functions of policies. The Bellman equation reads in vectorial form as:

$$V^{\pi} = R_{\pi} + \gamma \cdot P_{\pi} V^{\pi} \tag{2.5}$$

where R_{π} is the vector of rewards associated to each state by following policy π:

$$R_{\pi} = \begin{pmatrix} r(s_1, \pi(s_1)) \\ \vdots \\ r(s_{|\mathcal{S}|}, \pi(s_{|\mathcal{S}|})) \end{pmatrix}$$

and P_{π} is the transition matrix of policy π defined from the transition function of MDP :

$$P_{\pi} = \begin{pmatrix} p(s_1, \pi(s_1), s_1) & \cdots & p(s_1, \pi(s_1), s_{|\mathcal{S}|}) \\ \vdots & & \vdots \\ p(s_{|\mathcal{S}|}, \pi(s_{|\mathcal{S}|}), s_1) & \cdots & p(s_{|\mathcal{S}|}, \pi(s_{|\mathcal{S}|}), s_{|\mathcal{S}|}) \end{pmatrix}$$

Let the Bellman operator $\mathcal{T}_{\pi}$ defined as an operator on value vectors:

$$\mathcal{T}_{\pi} V = R_{\pi} + \gamma \cdot P_{\pi} V \tag{2.6}$$

Then Eq. (2.5) is rewritten: $V^\pi = \mathcal{T}_\pi V^\pi$. As the Bellman operator $\mathcal{T}_\pi$ is a contraction of factor γ w.r.t. the infinity norm ($||\mathcal{T}_\pi V - \mathcal{T}_\pi V'||_\infty \leq \gamma \cdot ||V - V'||_\infty$), V^π has unique fixed point satisfying $V^* = \mathcal{T}_\pi(V^*)$ [Bellman, 1986]. $\mathcal{T}_\pi$ is also a monotonous operator ($V \leq V' \Rightarrow \mathcal{T}_\pi V \leq \mathcal{T}_\pi V'$).

Let us define Π as the set of all possible policies, $V^*(s) = \max_{\pi \in \Pi} V^\pi(s)$ as the optimal value function of $s \in \mathcal{S}$, and $\pi^* = \arg\max_{\pi \in \Pi} V^\pi(s)$, then we have

$$V^{\pi^*}(s) = V^*(s) = \max_{\pi \in \Pi} V^\pi(s) \tag{2.7}$$

According to Bellman [1986], the optimal value function must verify the recursive Bellman optimality equation :

$$\forall s \in \mathcal{S}, V^*(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s,a,s') V^*(s') \right) \tag{2.8}$$

The Bellman optimality operator $\mathcal{T}$ for all value vectors V is then defined as:

$$\forall s \in \mathcal{S}, [\mathcal{T}V](s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s,a,s') V(s') \right) \tag{2.9}$$

Eq.(2.8) is written as: $V^* = \mathcal{T}V^*$. Like $\mathcal{T}_\pi$, the Bellman optimality operator $\mathcal{T}$ is a contraction of factor γ w.r.t. the infinity norm, and its unique fixed point is the optimal value function V^* [Bellman, 1986].

In the remainder of this work, we will use the notation $greedy(V)$ to represent the greedy policy w.r.t. V. If π is a greedy policy on V, we have $\mathcal{T}V = \mathcal{T}_\pi V$. Any optimal policy π^* can be defined as $greedy(V^*)$ [1].

On the other hand, the Bellman equations hold for the action value functions as well.

$$\forall (s,a) \in \mathcal{S} \times \mathcal{A}, \quad Q^\pi(s,a) = r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') Q^\pi(s', \pi(s')) \tag{2.10}$$

The above equation on Q-values can be presented in vectorial form:

$$Q^\pi = R + \gamma \cdot P'_\pi Q^\pi$$

where Q^π represents the Q-value functions in a vectorial form,

$$Q^\pi = \begin{pmatrix} Q^\pi(s_1, a_1) \\ Q^\pi(s_1, a_2) \\ \vdots \\ Q^\pi(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}) \end{pmatrix}$$

[1] Note that π^* is not necessarily unique.

R is the average rewards associated to all state-action pairs,

$$R = \begin{pmatrix} r(s_1, a_1) \\ r(s_1, a_2) \\ \vdots \\ r(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}) \end{pmatrix}$$

As the immediate reward function r no longer depends on the policy π, we use the notation R instead of R_π.

P'_π is the transition matrix between state-action pairs by following the policy π :

$$P'_\pi((s,a),(s',a')) = p(s,a,s') \cdot \mathbf{1}_{a'=\pi(s')}$$

where $\mathbf{1}_{a'=\pi(s')} = 1$ if $a' = \pi(s')$ and 0 otherwise. Note that P'_π is different from those used in the Bellman equations of the value functions.

The Bellman operator $\mathcal{T}'_\pi$ for any Q-value function is defined as:

$$\mathcal{T}'_\pi Q = R + \gamma \cdot P'_\pi Q$$

Like in the value function case, the Bellman optimality equation for Q-value functions reads:

$$Q^*(s,a) = r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s,a,s') \max_{a \in \mathcal{A}} Q^*(s',a')$$

and the Bellman optimality operator $\mathcal{T}'$ for Q-value functions is defined as:

$$[\mathcal{T}'Q](s,a) = r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s,a,s') \max_{a \in \mathcal{A}} Q(s',a')$$

As was shown by Bertsekas and Tsitsiklis [1995], there exists an equivalence between value functions V and action value functions Q, and the relationship between the two optimal value functions can be presented by the following equations:

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a) \tag{2.11}$$

$$Q^*(s,a) = r(s,a) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s,a,s') V^*(s') \tag{2.12}$$

In the remainder of this chapter, unless stated otherwise, the algorithms implemented on value functions V can also be implemented on the action value functions Q.

2.3.2 Value function based methods

Searching for the optimal policy by examining the return of each policy (brute force method) does not scale up. The main RL approaches define optimal policies based on learning of value functions.

2.3.2.1 Basic dynamic programming algorithms

A way to solve the Bellman equations and find the optimal value function is the use of dynamic programming (DP) methods such as value iteration and policy iteration [Bertsekas and Tsitsiklis, 1995].

Value iteration algorithm

Value iteration proceeds by directly computing the fixed point of the Bellman operation $\mathcal{T}$ (Eq.(2.9)), which is guaranteed to converge to the unique fixed point V^*. An optimal policy can be deduced from V^* by $greedy(V^*)$.

Algorithm 2.1: Value iteration

Input: stopping criterion $\epsilon > 0$
Output: approximated value function V_l verifying $||V^* - V_l||_\infty \leq \frac{\gamma}{1-\gamma}\epsilon$
Initialize $V_0 \leftarrow$ arbitrary initial values, $t \leftarrow 0$
repeat
  $V_{t+1} \leftarrow \mathcal{T}V_t$
  $t \leftarrow t + 1$
until $||V_t - V_{t-1}||_\infty < \epsilon$
return V_t

The value iteration algorithm is defined in Algorithm 2.1, which converges asymptotically towards the optimal value function V^*. In practice, it is not guaranteed that the convergence will be reached within a finite number of iterations, and the Bellman optimality operator $\mathcal{T}$ on V is applied iteratively until the infinity norm distance between the value functions of two successive iterations is less than ϵ, yielding a guaranteed upper bound of the distance between the optimal value function and the obtained value function [Bertsekas and Tsitsiklis, 1995]:

$$||V^* - V_t||_\infty \leq \frac{\gamma}{1-\gamma}\epsilon$$

The value function V^{π_t} of the greedy policy π_t derived from V_t ($\pi_t \leftarrow greedy(V_t)$) verifies that:

$$||V^* - V^{\pi_t}||_\infty \leq \frac{2 \cdot \gamma}{1-\gamma}\epsilon$$

Policy iteration

Policy iteration algorithm is presented in Algorithm 2.2. It proceeds as the following: firstly, one needs an initial policy π_0. Then this policy is evaluated by computing its correspondent value function (through the solution of Bellman equation). This step is noted as the *policy evaluation* step. Afterwards, one modifies the current policy by greedily choosing the updated V values. This step is called *policy improvement* step. The two steps

above are repeated until the stopping criterion is met (e.g. no effective policy improvement after policy evaluations).

In fact, the value function of policy π_t can be found by solving the Bellman equation (Eq. (2.5)) directly, because it is a linear system:

$$\begin{aligned} V^{\pi_t} &= r_{\pi_t} + \gamma \cdot P_{\pi_t} \cdot V^{\pi_t} \\ (I - \gamma \cdot P_{\pi_t}) \cdot V^{\pi_t} &= r_{\pi_t} \\ V^{\pi_t} &= (I - \gamma \cdot P_{\pi_t})^{-1} \cdot r_{\pi_t} \end{aligned}$$

However, computing the inverse of the $|\mathcal{S}| \times |\mathcal{S}|$ sized matrix $(I - \gamma \cdot P_{\pi_t})$ raises scalability problems, hence approximated approaches are used within policy evaluation step of Algorithm 2.2.

Policy iteration algorithm shares the convergence properties attributed to the value iteration algorithm [Szepesvári, 2010]. In general, the policy evaluation step is expensive in computation, but on the other hand, the policy iteration algorithm requires less iterations to converge [Bertsekas and Tsitsiklis, 1995]. Besides, policy iteration offers a guarantee of convergence to the optimal policy within a limited number of iterations. An intuitive illustration of the mechanism that value iteration and policy iteration algorithm search for the optimal value function is given in Figure 2.3.

Algorithm 2.2: Policy iteration algorithm

Policy iteration
Input: an initial policy π_{init}, and the stopping criterion of policy evaluation ϵ
Output: policy π_{t+1} verifying $||V^* - V^{\pi_{t+1}}||_\infty \leq \frac{2 \cdot \gamma}{1-\gamma}\epsilon$
Initialize $\pi_0 \leftarrow \pi_{init}$ and $t \leftarrow 0$
repeat
  $V_{t+1} \leftarrow Evaluate(\pi_t, \epsilon)$ // Policy evaluation
  $\pi_{t+1} \leftarrow greedy(V_{t+1})$ // Policy improvement
  $t \leftarrow t + 1$
until $\pi_t = \pi_{t+1}$
return π_{t+1}

Evaluate
Input: a policy π, and stopping criterion $\epsilon > 0$
Output: approximated value function $V^{\pi,l}$
Initialize $V^{\pi,0} \leftarrow 0$ and $l \leftarrow 0$
repeat
  $V^{\pi,l+1} \leftarrow \mathcal{T}^\pi V^{\pi,l}$
  $l \leftarrow l + 1$
until $||V^{\pi,l} - V^{\pi,l-1}|| < \epsilon$
return $V^{\pi,l}$

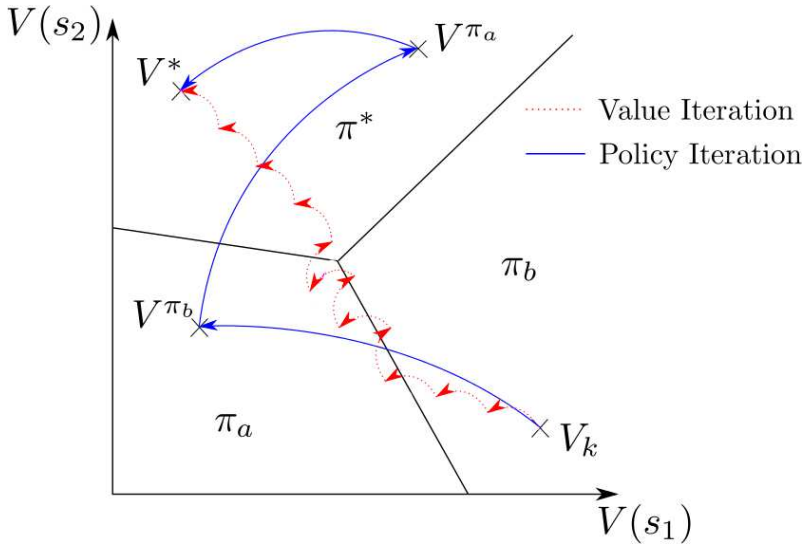


Figure 2.3: An intuitive illustration from [Thiéry, 2010] shows how value iteration and policy iteration algorithms search for the optimal value function. According to Bertsekas and Tsitsiklis [1995], the value space can be separated into several polyhedrons, each corresponding to a value range where some policy (π_a, π_b or π^*) is greedy. We suppose that the state space $\mathcal{S}$ contains only two states s_1 and s_2, and the value space is therefore a plane. In policy iteration, each the policy evaluation step searches for the value function $V^{\pi_{t+1}}$. Through several alternative policy evaluation and policy improvement steps, this algorithm reaches the optimal value. In value iteration, the search is realized by a set of small steps which approach the optimal value function progressively.

2.3.2.2 Approximated algorithms

Despite the optimality guarantees, the basic dynamic programming algorithms face two bottlenecks. The first bottleneck is the size of the problem. In reality, we often need to solve SDM problems the state and action spaces of which are large or continuous. In this case, the thorough exploration of value functions in the entire state/action space is intractable.

The second bottleneck is that the computational time needed to meet the stopping criterion defined by ϵ is unknown. The formal guarantee of the algorithm convergence to optimal value provided by [Sutton et al., 1999] relies on unrealistic assumptions, such as the sampling of all actions in all states.

We therefore need to estimate the value function based on function approximation methods. In practice, two types of RL algorithms based on function approximations are used to solve SDM problems with large or continuous state and action spaces : off-policy and on-policy algorithms. The off-policy algorithms may update estimated value functions on the basis of data provided externally (not acquired by executing a given policy). On-policy algorithms, on the other hand, update value functions strictly on the basis of experience gained from executing some policy. Off-policy and on-policy algorithms were respectively pioneered by Q-learning [Watkins and Dayan, 1992] and SARSA [Rummery and Niranjan, 1994].

Q-learning

Q-learning is an off-policy algorithm that approximates the optimal value function. This algorithm is composed of two parts:

- an update rule that, given (s, a, s', r) updates Q according to the transitions (s, a, s') and instant reward r;

- a sampling strategy to choose s and a, which can be determined from pre-generated training data, or obtained online by following some fixed policy π.

Algorithm 2.3 gives a formal description of Q-learning which requires a generative model of the considered problem. The policy π which chooses (s, a) pairs is usually implemented greedily w.r.t. the current Q-value function, possibly combined with $\epsilon-$greedy exploration. Singh et al. [2000] propose some other sampling strategies with theoretical guarantees.

Q-learning can also be implemented from data, without any generative model. In this case, for every (s, a, s', r) available in the data, the update function is applied. Watkins and Dayan [1992] proved the theoretical convergence of this algorithm, under the assumption that the sampling strategy asymptotically samples all actions, and that states and actions are discrete.

More generally, Q-learning can be combined with function approximation [van Hasselt, 2012], which makes it possible to apply the algorithm to larger problems, even when the state space is continuous.

Algorithm 2.3: Q-learning algorithm

Input: A generative model $\mathcal{M}$, a policy π, a learning rate $\alpha \in (0, 1]$
Output: approximated action value function Q
Initialize $Q(s, a)$ arbitrarily for all $(s, a) \in (\mathcal{S} \times \mathcal{A})$
repeat
  Select an initial state $s \in \mathcal{S}$
  repeat
    $a \leftarrow \pi(s)$
    $(s', r) \leftarrow \mathcal{M}(s, a)$
    $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s', a')]$
    $s \leftarrow s'$
  until s is the terminal state
until no more computational time
return Q

SARSA

SARSA is an on-policy algorithm which updates Q-value function based on experiences gained from taken actions. The difference between SARSA and Q-learning is in the update function : instead of taking optimal estimation of future Q-value, we simply choose the Q-value of the action a' taken according to the given policy π. Algorithm 2.4 provides a formal description of SARSA.

According to [Singh et al., 2000], this algorithm is guaranteed to converge to the optimal Q-value function Q^*, as long as all state-action pairs are visited an infinite number of times and the policy is gradually biased toward $greedy(Q^*)$ (defined in section 2.3.1).

Algorithm 2.4: SARSA algorithm

Input: A generative model $\mathcal{M}$, a policy π, a learning rate $\alpha \in (0, 1]$
Output: approximated action value function Q
Initialize $Q(s, a)$ arbitrarily for all $(s, a) \in (\mathcal{S} \times \mathcal{A})$
repeat
  Select an initial state $s \in \mathcal{S}$
  repeat
    $a \leftarrow \pi(s)$
    $(s', r) \leftarrow \mathcal{M}(s, a)$
    $a' \leftarrow \pi(s')$
    $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \cdot Q(s', a')]$
    $s \leftarrow s'$
  until stopping criterion is met
until no more computational time
return Q

2.4 Direct policy search

The RL approaches are based on defining the optimal value functions on each state or each state-action pair. Direct policy search, quite the contrary, associates a score or fitness to each policy π, and explores a subset of the policy space according to the fitness information. In its simplest form, the quality of π is defined as a fitness function $\rho(\pi)$, which brings a total order among all policies.

In the real world problems, one most usually considers the policy space as a parameter space, e.g. the weight vectors of a (fixed topology) neural networks [Hornik et al., 1989]. Let π_θ denote the policy associated to a parameter vector θ, then the fitness function $\rho(\pi_\theta) = \rho(\theta)$. When the function ρ is differentiable w.r.t. θ, one can use gradient descent method [Snyman, 2005] to search for the optimal policy. Since an analytic expression for the gradient is not always available, one must rely on an estimation of the gradient. Despite the progress made [Stulp and Sigaud, 2012], the drawback of gradient based methods is proven to fall in local optima, entailing two weaknesses – instability of performance and poor reproducibility of results.

In the case where ρ is not differentiable w.r.t. θ, gradient-free methods, such as simulated annealing [Kirkpatrick, 1984], cross-entropy search [Rubinstein and Kroese, 2004] and evolutionary computation [Fogel, 2006], have also been intensively used to solve SDM problems. However, the gradient-free methods face a bottleneck. The fitness function defines a noisy optimization problem, as the fitness function $\rho(\pi)$ is usually defined as an expectation of the policy return over a distribution of the starting positions of the studied model. An approximation of the fitness function must thus be used to decrease the computational cost through, for instance, Bernstein races to prune the unpromising solutions [Heidrich-Meisner and Igel, 2009] or through surrogate optimization basis [Loshchilov, 2013].

2.5 Exploration vs. exploitation dilemma

Providing the guarantee of finding the optimal policies in the sense of expected cumulative reward, mainstream RL algorithms face the problem of scalability due to their thorough exploration of the state and action spaces. For many applications with focus on on-line performance, the thorough exploration of the state and action spaces is not feasible. An alternative approach is based on the multi-armed bandit (MAB) algorithms, from the game theory literature [Auer et al., 2002].

The term *bandit* refers to the name of a slot name (one-armed bandit) in the casinos. In an MAB problem, a player faces a finite number of independent slot machines (or arms). Each machine has a fixed unknown expected return. The player iteratively selects a machine (pull an arm). Since the player wishes to earn as much reward as possible, the choice of which machine to play should enforce a balance between the exploration and exploitation (the EvE trade-off): the player must decide whether to pull the most rewarding arms according to the past observations (exploitation), or to pull arms that were not pulled frequently enough (exploration), since those arms whose reward were

not high could have been underestimated if the first pulls were unlucky. The MAB thus define a decision making problem based on a one-state MDP, which can be extended to a sequential decision making problem, referred to as MCTS (section 2.6.1). In the following, formal definition and some theoretical results of the multi-armed bandit problem will be presented.

2.5.1 MAB settings

Let us define the multi-armed bandit problem with a finite number K of arms noted as $\mathcal{A} = \{1, 2, \cdots, K\}$. In its original formulation [Robbins, 1985], each arm $a \in \mathcal{A}$ corresponds to a probability distribution P_a on $[0, 1]$. At each time step $t \in \mathbb{N}$, the player selects (or pulls) arm a_t, and then receives a random reward r_{a_t} drawn from the distribution P_{a_t}. Setting the time horizon to T, the player's objective in the MAB problem is to maximize the sum of rewards $\sum_{t=1}^{T} r_{a_t}$. For each arm $a \in \mathcal{A}$, let n_a denote the number of times a has been selected, and $r_{a,i}$ denote the reward received by arm a at the i-th time. Then for any $a \in \mathcal{A}$, $r_a = \{r_{a,i} | i = 1, \cdots, n_a\}$ is an i.i.d. sample set drawn after the distribution P_a.

It is natural to define the expected reward, and the empirical mean reward of arm a when it is played for the n_a-th time:

$$\mu_a = \mathbb{E}(r_a) \tag{2.13}$$

$$\hat{r}_a = \frac{1}{n_a} \sum_{i=1}^{n_a} r_{a,i} \tag{2.14}$$

with $\hat{r}_a \xrightarrow{a.s.} \mu_a$ when $n_a \to \infty$.

Let us describe the decision rule of the player by strategy π, which maps the history of previous arm selections and rewards received to the next arm to pull. For simplicity reason, let us denote π_{MAB} as a mapping from the current time step number t to the next chosen arm: $\pi_{MAB} : \mathbb{N} \to \mathcal{A}$. The MAB framework can be summarized by Algorithm 2.5.

Algorithm 2.5: Multi-armed bandit framework

Known parameters: number of arms K, time horizon T with $T \geq K \geq 2$.
Unknown parameters: K probability distributions $\{P_1, P_2, \cdots, P_K\}$.
for all $t = 1, 2, \cdots, T$ **do**
  Select $a_t = \pi_{MAB}(t)$ based on the $\hat{r}_{a_t}, n_{a_t}$ information available at time $t - 1$
  Draws the reward $r_{a_t} \sim P_{a_t}$
  $n_{a_t} \leftarrow n_{a_t} + 1$
  $\hat{r}_{a_t} = \frac{1}{n_{a_t}} \sum_{i=1}^{n_{a_t}} r_{a_t,i}$
end for
Goal: Maximize the cumulative gain $\sum_{t=1}^{T} r_{a_t}$

2.5.2 Optimality criteria

Several optimality criteria are used in MAB solutions.

Cumulative regret

Let us note the best expected reward by $\mu^* = \max_{a \in \mathcal{A}} \mu_a$, and the margin (or regret) of each arm a is measured by $\Delta_a = \mu^* - \mu_a$. The expected cumulative regret in a multi-armed bandit problem at time step t is defined as

$$\Omega_t = \mathbb{E}(t \cdot \mu^* - \sum_{i=1}^{t} r_{a_t}) = t \cdot \mu^* - \sum_{a=1}^{K} n_a \mu_a = \sum_{a=1}^{K} n_a \Delta_a \tag{2.15}$$

The maximization of cumulative gain in MAB problem is thus equivalent to the minimization of the regrets.

Simple regret

The simple regret in a multi-armed bandit problem at time step t is defined as

$$\omega_t = \mu^* - \mathbb{E}(r_{a_t}) = \Delta_{a_t} \tag{2.16}$$

In many situations, an arm is chosen for the pursuit of instant reward instead of cumulative rewards. For example, after a trail period of several products, one company decides to commercialize one product with the best quality. In this case, what matters is the performance of the single best product, rather than the cumulative rewards of all products obtained in the trial phase. The simple regret is used as the optimality criterion in these situations.

2.5.3 MAB Algorithms

Multiple MAB strategies have been devised in the literature, depending on the player's objective.

Random Uniform

The most straightforward strategy is the random uniform selection of arms – picking each arm $a \in \mathcal{A}$ with probability $1/|\mathcal{A}|$ in each time step. Although such choice is not optimal for the minimization of cumulative regret, it has been shown by [Bubeck et al., 2009] that random uniform selection of arms is an optimal strategy for minimizing the simple regret asymptotically: the value $\mathbb{E}(r_t)$ gradually converges to the optimal value when $t \to \infty$ if the empirical best arm were selected after t rounds of uniform selection.

ϵ-greedy

ϵ-greedy is the first widely used MAB strategy, in which the EvE trade-off is controlled by the parameter $\epsilon \in]0, 1[$: choosing the arm with empirical best mean reward with probability $1 - \epsilon$, and uniformly randomly picking other arms with probability ϵ. This strategy offers a better cumulative regret than the random uniform strategy.

In order to achieve better asymptotic expected cumulative regret, ϵ_t-greedy strategies have been proposed, where $\epsilon_t \to 0$ when the time horizon $t \to \infty$. By carefully choosing ϵ_t, a cumulative regret in the order of $\mathcal{O}(\log t)$ can be obtained [Auer et al., 2002]. However, the best design of ϵ_t requires the knowledge about the distribution of rewards of the arms, which is not always available.

Upper Confidence Bound (UCB)

Proposed by Auer et al. [2002], UCB is a method that considers the expectations and variances of arm values at the same time. It requires that each arm should be pulled for at least once. Then upper confidence bounds on the rewards of each arm are computed at each time step, and the arm with the largest upper bound will be chosen. The simplest and most implemented UCB policy is as follows.

Definition 1. *(UCB1) The UCB1 strategy is the strategy that firstly pulls every arm once, and then selects at round $t > K$ an arm $a \in \{1, 2, \cdots, K\}$ that maximizes*

$$UCB1_t(a) = \hat{r}_a + \sqrt{\frac{2 \ln t}{n_a}} \tag{2.17}$$

The first (respectively the second) term on the R.H.S. of Eq.(2.17) corresponds to the exploitation (resp. exploration) term. It has been proved that the upper bound of the cumulative regret obtained by the UCB1 strategy grows logarithmically with the number of total number of arm pulls t [Auer et al., 2002].

Beside UCB1, other upper confidence bound estimates have been proposed, such as UCB1-Tuned. Let us define the upper bound on the variance of reward estimates as

$$V_a(T) = \frac{1}{T} \sum_{t=1}^{T} r_{a_t}^2 - \mu_a^2 + \sqrt{\frac{2 \ln T}{n_a}} \tag{2.18}$$

Then

$$UCB_1Tuned_t(a) = \hat{r}_a + \sqrt{\frac{2 \ln t}{n_a} \min\{1/4, V_a(t)\}} \tag{2.19}$$

Compared to UCB1, UCB1-Tuned has a refined estimate of the upper bound of arm values. Despite the same theoretical upper bound on cumulative regrets [Auer et al., 2002], UCB1-Tuned has been shown to perform substantially better than UCB1.

Figure 2.4: The search tree and three phases of the MCTS algorithm.

2.6 Monte-Carlo Tree Search

Monte-Carlo tree search (MCTS) extends MAB to tree structured search space [Coulom, 2006]. Recently, MCTS, including the famed Upper Confidence Tree (UCT) algorithm [Kocsis and Szepesvári, 2006] and its variants, has been intensively investigated to handle sequential decision problems. MCTS, notably illustrated in the domain of Computer-Go [Gelly and Silver, 2007], has been shown to efficiently handle medium-size state and action search spaces through a careful balance between the exploration of the search space, and the exploitation of the best results found so far. While providing some consistency guarantees [Berthier et al., 2010], MCTS has demonstrated its merits and wide applicability in the domain of games [Ciancarini and Favini, 2009] or planning [Nakhost and Müller, 2009] among many others.

2.6.1 MCTS algorithm

In this section, We present the MCTS framework, referring the readers to [Gelly and Silver, 2007; Chaslot et al., 2008a] for complementary presentations.

In an SDM problem, given a state s, if we have access to the action value functions $Q(s,a)$, then the optimal policy π^* can be generated from Q : $\pi^*(s) = \arg\max_a Q(s,a)$. The MCTS algorithm approximates $Q(s,a)$ values through simulation of trajectories starting with (s,a). In general, uniform simulations do not give correct estimations of action values. Only by biasing the trajectories towards an optimal behaviour, average rewards could converge to the Q^*. Such biased trajectories are achieved by gradually constructing an unbalanced tree, in which nodes represent visited states, and branches represent actions. Favouring the most promising nodes, when more simulations are made, more simulations will be made on promising actions, thus giving them more weight on the average reward computation. As the number of simulation grows, one can expect that the average reward obtained over these simulations would give a good estimation of $Q^*(s,a)$.

MCTS simultaneously explores and builds a search tree, initially restricted to its root node, along N tree-walks a.k.a. simulations. As is illustrated in Figure 2.4, each tree walk

involves three phases:

The selection phase Each tree-walk starts from the root node and iteratively selects an action/a child node until arriving in a leaf node. In the best known MCTS algorithm – Upper Confidence Bounds applied to Trees (UCT) [Kocsis and Szepesvári, 2006], the action selection is handled as a multi-armed bandit problem. The set $\mathcal{A}_s$ of admissible actions a defines the possible child nodes (s, a) of node s; the selection of action a^* maximizes the Upper Confidence Bound:

$$\bar{r}_{s,a} = \hat{r}_{s,a} + \sqrt{c_e \ln(n_s)/n_{s,a}} \tag{2.20}$$

over a ranging in $\mathcal{A}_s$, where n_s stands for the number of times node s has been visited, $n_{s,a}$ denotes the number of times a has been selected, and $\hat{r}_{s,a}$ is the average reward collected when selecting action a from node s. The difference between Eq.(2.20) and Eq.(2.17) (UCB1) is that the EvE tradeoff is controlled by parameter c_e.

Upon the selection of a^*, the next state is drawn from the transition function depending on the current state and a^*. In the remainder of this manuscript, a tree node is labelled with the sequence of actions followed from the root; the associated reward is the average reward collected over all tree-walks involving this node.

The tree building phase Upon arriving in a node s, some action $a \in \mathcal{A}_s$ is (uniformly or heuristically) selected and (s, a) is added as child node of s. Accordingly, the number of nodes in the tree is the number of tree-walks.

The random phase Starting from a leaf node (s, a), actions are iteratively selected according to a default policy, often set to the uniform policy or a domain specific one, until arriving in a terminal state u. With the generative model of the problem, the total reward r_u of the whole tree walk is computed and used to update the statistics in all nodes (s, a) visited during the tree-walk:

$$\begin{aligned} \hat{r}_{s,a} &\leftarrow \frac{1}{n_{s,a}+1}(n_{s,a} \times \hat{r}_{s,a} + r_u) \\ n_{s,a} &\leftarrow n_{s,a} + 1; \qquad n_s \leftarrow n_s + 1 \end{aligned} \tag{2.21}$$

Algorithm 2.6 gives a formal description of MCTS.

2.6.2 MCTS extensions

Besides its celebrated application to Computer Go [Gelly and Silver, 2007], MCTS has been extended to many other SDM problems with large search space, often through the use of additional heuristics.

Algorithm 2.6: MCTS algorithm

Monte-Carlo tree search
Input: number N of tree-walks
Output: search tree $\mathcal{T}$
Initialize $\mathcal{T} \leftarrow$ root node (initial state)
for $t = 1$ **to** N **do**
  TreeWalk($\mathcal{T}$, root node)
end for
return $\mathcal{T}$

TreeWalk
Input: search tree $\mathcal{T}$, node s
Output: reward r_u
$\mathcal{A}_s = \{$ admissible actions not yet visited in $s\}$
if $\mathcal{A}_s = \emptyset$ **then**
  // Selection phase
  Select $a^* = \text{argmax}\ \{\bar{r}_{s,a}, (s,a) \in \mathcal{T}\}$ // Eq.2.20
  $r_u \leftarrow \text{TreeWalk}(\mathcal{T}, (s, a^*))$
else
  // Tree building phase
  Select a^* uniformly from $\mathcal{A}_s$
  Add (s, a^*) as child node of s
  // Random phase
  $r_u \leftarrow \text{RandomWalk}((s, a^*))$
end if
Update n_s, n_{s,a^*} and $\hat{r}_{s,a^*}$ // Eq. (2.21)
return r_u

RandomWalk
Input: state u
Output: reward of the simulation r_u
while u is not final state **do**
  Uniformly select an admissible action a for u
  $u \leftarrow (u, a)$
end while
$r_u = evaluate(u)$ //calculate the reward of the entire tree-walk
return r_u

Many armed bandit

In order to prevent over-exploration when the number of admissible arms is large w.r.t. the number of simulations (the so called many armed bandit issue [Wang et al., 2008]), the Progressive Widening (PW) heuristics [Coulom, 2006] has been used in many MCTS variants. In PW, the allowed number of child nodes of node s is initialized to 1, and increases with its number of visits n_s like ${n_s}^{1/b}$ (with b usually set to 2 or 4). Such heuristics favours building deeper trees.

In the many armed bandit problem, since only a subset of arms will be visited within a limited number of simulations, the choice of arms to be pulled first should be done carefully. For the sake of convergence speed, it is clearly desirable to consider the best options as early as possible. The RAVE heuristic [Gelly and Silver, 2007] aims at exploring earlier the most promising regions of the search space. In its simplest version, RAVE(a) is set to the average reward taken over all tree-walks involving action a. The RAVE vector can be used to guide the tree-building phase, that is, when selecting a first child node upon arriving in a leaf node s, or when PW heuristic is triggered and a new child node is added to the current node s. In both cases, the selected action is the one maximizing RAVE(a).

Macro-actions

In SDM problems, the size of the search space Π grows exponentially with the policy length. Suppose that the policy length is l, and branching factor in each step of the policy is b, then the total number of policies in the search space is b^l.

Grouping a sequence of actions as one macro-action can reduce the effective branching factor of the search tree. If we define a macro-action A as a sequence of actions $(a_1, a_2, \cdots, a_M)$, and limit the number of macro-actions as B ($B \leq b^M$), then the size of the search space based on macro-actions will be $B^{l/M}$ satisfying $B^{l/M} \leq b^l$. The macro-action length parameter M controls the tradeoff between the granularity of possible strategies and the forward planning potential of the search tree. Experiments on artificial game trees [Childs et al., 2008] and the physical travelling salesman problem [Powley et al., 2012] have demonstrated the merits of search based on macro-actions.

Partially observable games

Previous applications of MCTS are mostly based on perfect information models, in which the agent has access to the state s identifying the stationary reward distributions. Being the model and testbed for many real world SDM problems, games with incomplete information – also called Partially Observable Games (POGs) – are games where players know the rules but cannot fully see the actions of other players and real state of the game, e.g. card games. The most well known partially observable games include poker [Ponsen et al., 2010], Kriegspiel [Ciancarini and Favini, 2009] and phantom-go [Cazenave, 2006].

Studies on the partially observable games by using MCTS have begun in recent years. Ciancarini and Favini [2009] show that in the incomplete information setting, simulating the whole game (until the terminal state) in MCTS usually results in a worse estimate

of action values than simulating just a limited number of steps, because the latter one introduces less noise in the state evaluation. In other words, in the incomplete information settings, spending much effort to search for the long term strategies is less helpful than focusing on the search of short term strategies. Auger [2011] introduces a multiple-tree technique, in which both the behavior of the player and its opponents are modelled by search trees. Through the construction of opponent's search trees, the proposed algorithm makes better predictions on the opponents behavior and achieves a better evaluation of the action values.

Continuous planning

Motivated by applications in economics (investment plan), engineer (water level control for hydro power dams) and robotics (wheel speed control), the MCTS framework has been extended to handle SDM problems with continuous state and action spaces.

Most current results in field of continuous planning with MCTS variants rely on the discretization of the action space [Auer et al., 2007; Mansley et al., 2011; Weinstein and Littman, 2012]. Through the use of Gaussian convolution-based smoothing, [Couetoux et al., 2011] proposed an extension of the RAVE heuristic which allows the estimation of state and action values in the continuous space based on samples.

Knowing that all actions can not be tried within a limited time in the continuous action space, in order to keep the consistency of MCTS, a stochastic tree building strategy (Double Progressive Widening) which favours selecting actions from intervals in which better rewards were obtained in past, has been proposed in [Couëtoux et al., 2011; Auger et al., 2013].

[Couëtoux et al., 2012] provides another possibility called Blind Value (BV) which helps the exploration of new actions. The idea of BV is to try actions that are far away from known actions during the first simulations, and then to focus on areas that have many actions with high $\bar{r}_{s,a}$ values.

Parallelization

The scalability of MCTS, i.e. its ability to generate better policies when additional computational power is provided, has been praised as a major advantage of MCTS. However, as is shown by the experiments on Computer Go [Bourki et al., 2011], although the performance (success rate) of MCTS improves when more computational power is allocated, the improvement speed follows a diminishing return law.

In the real-time SDM problems, as the computational cost per step (also called *reflection time*)is fixed, the parallelization is a principal way to improve the MCTS performance. In the recent literature [Bourki et al., 2011], the MCTS algorithm has been parallelized through three message-passing methods (in which communications are explicit and limited) over multi-core machines (clusters):

- *Fast tree parallelization* consists of carrying out multiple random simulations on different cores while keeping only one tree in the memory of a master machine. This

method is expensive in terms of communication especially when RAVE values are used [Gelly et al., 2008].

- *Slow tree parallelization* consists of having one tree on each computation node, and to synchronize these trees slowly, i.e. not at each simulation but with a certain frequency such as three times per second [Gelly et al., 2008].
- *Very slow root parallelization* is a special case of slow tree parallelization, but on the lowest possible frequency $f = 1/t$ with t the allowed reflection time per step. The trees are only synchronized once at the end of the reflection time in each decision step, and the drawback is that there is no load balancing between computational nodes.

Current experimental results show that the slow tree parallelization method outperforms the other two methods and represents the state of the art in MCTS parallelization [Bourki et al., 2011].

Chapter 3

Multi-Objective Optimization

In this chapter, we firstly describe the formal background of multi-objective optimization (MOO), and discuss the critical issues in this domain. Then the chapter reviews the state of the art in MOO.

3.1 MOO formal background

3.1.1 Problem statement

The simultaneous optimization of two or more conflicting objectives is called multi-objective optimization (MOO). Practical optimization problems, especially the engineering design optimization problems, often have a multi-objective nature. For example, in the engineering design problem, some structural performance criteria are to be maximized, while the weight of the structure and the implementation costs should be minimized simultaneously.

An MOO problem with n decision variables $(x_1, x_2, \cdots, x_n)$ and d objectives $(f_1, f_2, \cdots, f_d)$ is formulated as follows:

$$\begin{aligned} Optimize \quad & \mathbf{y} = f(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_d(\mathbf{x})) \\ s.t. \quad & \mathbf{x} = (x_1, \cdots, x_n) \in \mathcal{X} \end{aligned} \tag{3.1}$$

where the decision vector $\mathbf{x}$ ranges in the decision (parameter) space $\mathcal{X}$, $\mathbf{y}$ is the objective vector with f_i mapping $\mathcal{X}$ onto $\mathbb{R}$, $\mathbb{R}^d$ is the objective space. The objective function is the mapping $f : \mathcal{X} \to \mathbb{R}^d$. In the following, without loss of generality, we only consider objectives $f_i, i = 1, 2, \cdots, d$ to be maximized.

3.1.2 MOO optimality

There are many ways to formulate an MOO problem. The key question in MOO regards the trade-off between the conflicting objectives $f_i, i = 1, 2, \cdots, d$. This section concentrates on the concept of Pareto optimization at the core of multi-objective optimization, originated from the engineer/economist Vilfredo Pareto [Pareto, 1896], stating that:

Multiple criteria solutions could be partially ordered without making any preference choices a priori.

Several notions related to Pareto optimality are frequently used in the MOO literature as the following, referring the reader to [Deb, 2001] for more detail.

Definition 2. *(**Weak Pareto dominance**) Given two objective vectors* $\mathbf{y} = (y_1, \ldots, y_d), \mathbf{y}' = (y'_1, \ldots, y'_d)$, $\mathbf{y}$ *is said to weakly dominate* $\mathbf{y}'$ *(noted* $\mathbf{y} \succeq \mathbf{y}'$*) iff* $y_i \geq y'_i$ *for all* $i = 1, \cdots, d$.

Definition 3. *(**Pareto dominance**) Objective vector* $\mathbf{y}$ *dominates objective vector* $\mathbf{y}'$ *(noted* $\mathbf{y} \succ \mathbf{y}'$*) if* $\mathbf{y} \succeq \mathbf{y}'$ *and* $y_i > y'_i$ *for at least one* $i \in \{1, 2, \cdots, d\}$.

Definition 4. *(**Incomparability of vectors**) Objective vectors* $\mathbf{y}$ *and* $\mathbf{y}'$ *are incomparable (noted* $\mathbf{y} \parallel \mathbf{y}'$*) iff* $\mathbf{y} \not\succeq \mathbf{y}'$ *and* $\mathbf{y}' \not\succeq \mathbf{y}$.

Definition 5. *(**Pareto optimality**) The solution* $\mathbf{x}^*$ *and its correspondent objective vector* $f(\mathbf{x}^*)$ *are Pareto optimal iff* $\nexists \mathbf{x} \in \mathcal{X}$ *such that* $f(\mathbf{x}) \succ f(\mathbf{x}^*)$.

For the sake of simplicity, we will interchangeably speak of Pareto dominance for the decision vector $\mathbf{x}$ and the associated objective vector $f(\mathbf{x})$ in the remainder of this manuscript.

Definition 6. *Given a point set* P, P^* *is the set of points in* P *which are non-dominated by points in* P*, referred to as Pareto front w.r.t.* P.

$$P^* = \{\mathbf{y} \in P : \nexists \mathbf{y}' \in P \ s.t \ \mathbf{y}' \succ \mathbf{y}\}$$

Definition 7. *Point set* P *is called a non-dominated set iff* $P^* = P$.

Definition 8. P^{**} *is the optimal Pareto front in the considered MOO problem if it includes all points which are non-dominated by other points in* $\mathcal{X}$.

Definition 9. *(**Comparison between non-dominated sets**) A non-dominated set* P_1 *is said to be better than another non-dominated set* P_2 *(noted* $P_1 \rhd P_2$*) iff every* $\mathbf{y} \in P_2$ *is weakly dominated by at least one* $\mathbf{y}' \in P_1$ *and* $P_1 \neq P_2$.

Having the relationship between non-dominated sets, points in P are further ordered through the *Pareto rank* function [Deb et al., 2000].

Definition 10. *The Pareto ranks w.r.t. a set of objective vectors* $P \subseteq \mathbb{R}^d$ *are determined in an iterative manner as follows: all non-dominated points in* P *(noted as* P^* *or* $\mathcal{F}_1(P)$*) are given rank 1. The set* $\mathcal{F}_1(P)$ *is then removed from* P*; from the reduced set, the non-dominated set are given rank 2 (noted as* $\mathcal{F}_2(P)$*); the process continues until all points of* P *have received a Pareto rank. The Pareto rank of a point* $p \in P$ *is denoted* $i_{rank}(p, P)$ *(Figure 3.1).*

Noting the largest Pareto rank of points in P by $i_{rank;max}(P)$, we have by construction $\forall i, j \in \{1, \ldots, i_{rank;max}(P)\}, i < j \Rightarrow \mathcal{F}_i(P) \rhd \mathcal{F}_j(P)$.

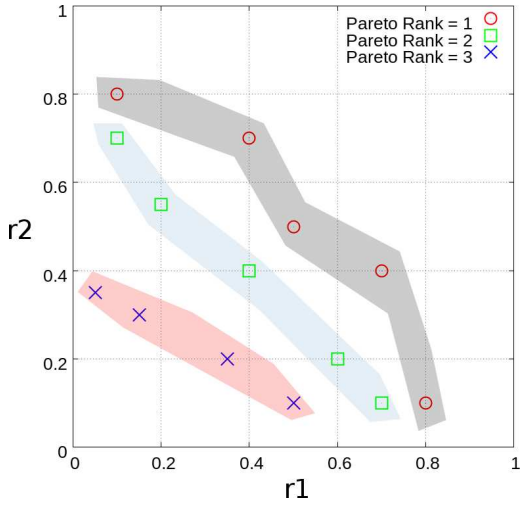


Figure 3.1: Three non-dominated sets partitioned according to their Pareto ranks.

3.1.3 An MOO example

We use the following 2-variable bi-objective optimization problem to illustrate some of the above notions in MOO:

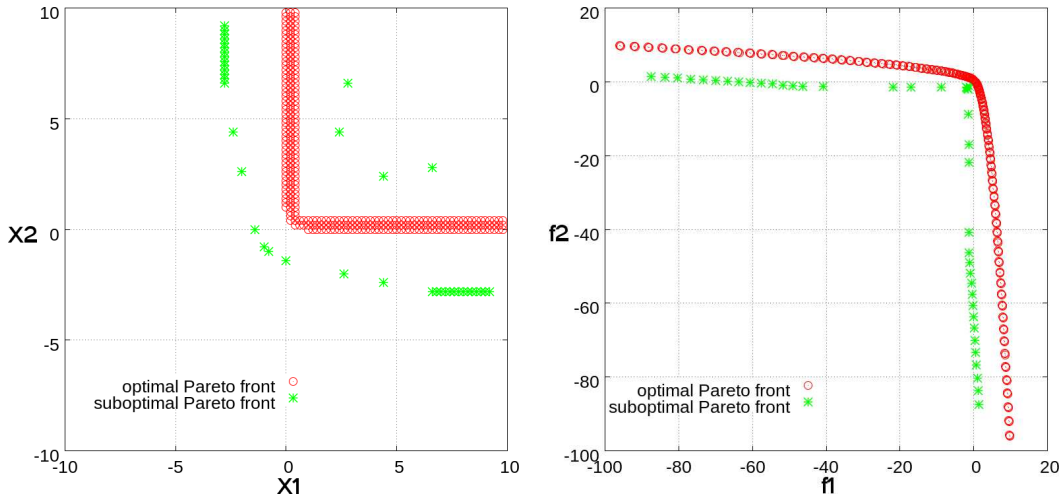
$$\begin{aligned} Maximize \quad & f_1(x_1, x_2) = x_1^2 + x_2 \\ & f_2(x_1, x_2) = x_1 + x_2^2 \\ s.t. \quad & -10 \leq x_1 \leq 10 \\ & -10 \leq x_2 \leq 10 \end{aligned} \tag{3.2}$$

Figure 3.1.3 illustrates a set of 280 Pareto optimal solutions for this problem (in red) and one set of suboptimal solutions (in green). The optimal solutions form the *Pareto front* in the objective space. Note that the Pareto front in the decision space is neither necessarily convex, nor continuous.

3.1.4 MOO critical issues

In order to compare two MOO solution sets, the standard procedure is to compare their Pareto optimal set. The difficulty lies in the fact that there exists no total order among solution sets. In the single-objective context, the solution sets can be compared according to their optimal elements. In the multi-objective case, however, such total order does not exist, and the the Pareto front of two solution sets might be incomparable (Figure 3.3). As noted by [Deb, 2001], the issue of incomparable solution sets becomes even more severe as the number of objectives d increases.

Comparing two solution sets is an MOO problem per se. Still, we need a total order according to some preference information to assess rigorously the solution sets in MOO algorithms. To do this, a widely accepted procedure is to use *quality indicators*, suggested by Zitzler et al. [2001] and Knowles et al. [2006]. In section 3.3.3, three quality indicators,



(a) Decision Space $\mathcal{X}$ (b) Objective Space $\mathbb{R}^2$

Figure 3.2: A set of solutions (Left) and their correspondent objective values (Right) for the bi-objective example problem (Eq. 3.2), among which the optimal Pareto front is marked by red circles. The red points belong to the optimal Pareto front. The set of green points is non-dominated in the sense of Definition 7, although these solutions are dominated by the red ones.

respectively the generational distance (GD), inverted generational distance (IGD) [Zitzler et al., 2003] and hypervolume indicator [Zitzler and Thiele, 1998] will be defined to measure the quality of non-dominated solution sets, regarding 1) the distance between the non-dominated solutions P^* and the Pareto optimal solutions P^{**} to be minimized; 2) the diversity of points within P^* to be maximized.

3.2 Classification of MOO approaches

Two MOO approaches – the preference based and ideal approaches – are distinguished in the literature, depending on whether the user is required to express his/her preference before starting search [Deb et al., 2000], as is illustrated in Figure 3.4.

In the preference based approaches (Figure 3.4-Left), the user defines a priori his/her preferences[1] on the objective vectors, then the MOO problem is handled as a (series of) single-objective problem(s) which searches for the solution that best satisfies the user preference.

The ideal approaches suggest to first find all Pareto optimal solutions of the MOO problem (Figure 3.4-Right), then choose a posteriori one or several solutions taking into account the preference information.

[1]The user's preferences can be considered as an abstract utility function in the mind of the decision maker faithfully reflecting his/her preferences (Figure 3.5(a)) [Marler and Arora, 2010].

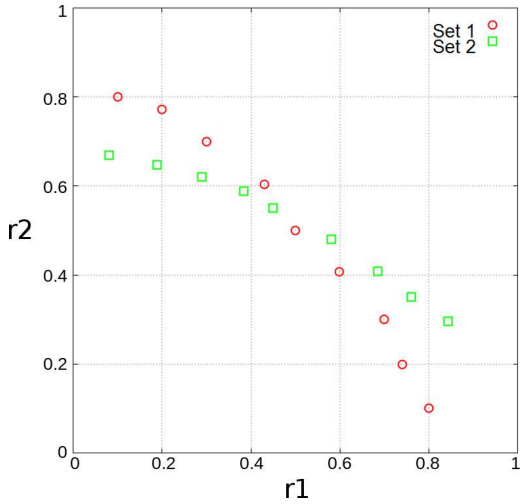


Figure 3.3: Two sets of non-dominated solutions may be incomparable in the multi-objective space.

The user preference is explicitly presented in the form of a scalarization function $g_\rho(f(\mathbf{x})) : \mathbb{R}^d \rightarrow \mathbb{R}$ in the preference based approaches, where ρ is the parameter representing different user preferences. The main advantage of the preference based approaches is that the solution can be found using one of the many single-objective optimizers available in the optimization literature. Some of the most popular preference based MOO approaches are presented as the following.

3.2.1 Weighted sum method

Due to its simplicity, the weighted sum method is one of the most popular preference based approaches for solving MOO problems. It has been used extensively to provide a single solution point that reflects preferences presumably incorporated in the weight settings. In weighted sum methods, the optimization of multiple objective functions $(f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_d(\mathbf{x}))$ is reformulated as the maximization of a scalar utility function:

$$\begin{aligned} Maximize \quad & g_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{d} w_i \cdot f_i(\mathbf{x}) \\ s.t. \quad & \mathbf{x} = (x_1, \cdots, x_n) \in \mathcal{X} \end{aligned} \tag{3.3}$$

where $w_i \geq 0, i = 1, 2, \cdots, d$ and $\sum_{i=1}^{d} w_i = 1$.

In principle, any single objective optimization method can be applied for solving the above problem. Note that choosing the values for weights defines the relative importance of each objective function, thus reflecting the user preferences.

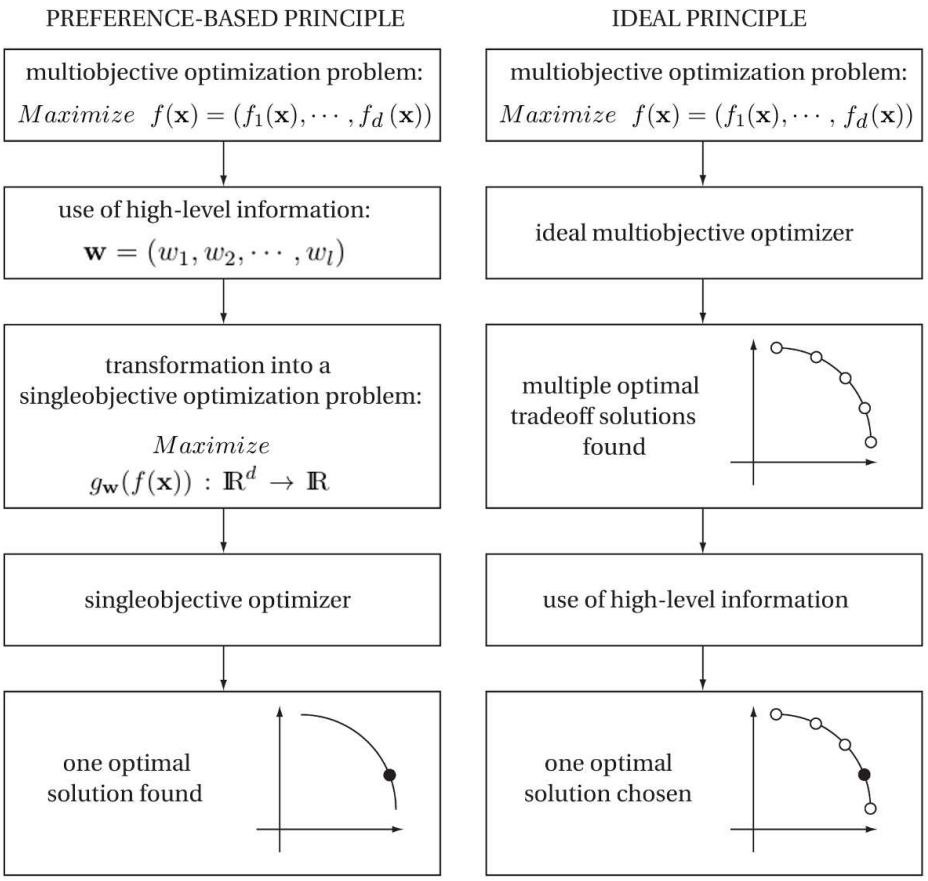


Figure 3.4: Left: the preference based approach of MOO. Note that the number of hyper-parameters ($|\mathbf{w}| = l$) in the user preference function is not necessarily the same as the number of objectives d. Right: the ideal approach of MOO. This illustration is adapted from [Tušar, 2007].

Convex Pareto front case

The maximization of Eq. (3.3) provides a sufficient condition for Pareto optimality, which means the maximum of $g_{\mathbf{w}}(\mathbf{x})$ is always Pareto optimal [Zadeh, 1963]. Lin [1976] further shows that when the Pareto optimal hypersurface of the objective space is convex, the entire Pareto optimal solution set can be obtained by the weighted sum method through the consistent variation of weight settings.

In nature, the weight settings in the weighted sum method can be considered as a linear approximation of the preference function . The isolines of the utility function $g_{\mathbf{w}}(\mathbf{x})$ approximate the isolines of the abstract preference function (Figure 3.2.1 (a) and (b)), which intersects with the Pareto optimal point in the objective space that brings the most satisfaction to the user.

Despite its simplicity and optimality guarantee in the convex Pareto front case, the weighted sum method faces one bottleneck. For multi-objective problems in practice, there are often infinitely many Pareto optimal solutions, especially in the problems with continuous objective space, thus it is often required to search for an approximated set of Pareto optimal solutions which gives an evenly distributed coverage of the Pareto front. However, Marler and Arora [2010] point out that, evenly distributed choices of weight settings does not necessarily lead to an evenly distributed Pareto optimal set. The choice of weight settings for an appropriate coverage of the Pareto front remains a challenging issue.

Non-convex Pareto front case

Being a linear-scalarization method, the main disadvantage of the weighted sum method is that it can not find solutions which lie in the non-convex regions of the Pareto front [Deb, 2001], an example of such solution is shown in Figure 3.5(c).

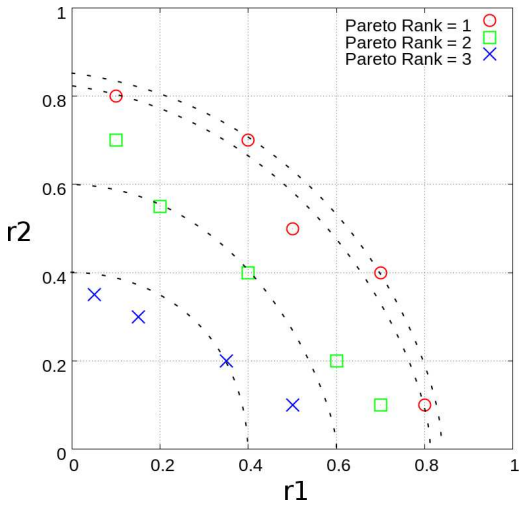
3.2.2 ϵ-constraint method

The ϵ-constraint method was first proposed by Marglin [1967]. In this method, only one objective function is maximized, while others are subject to constraints defined by the ϵ parameter:

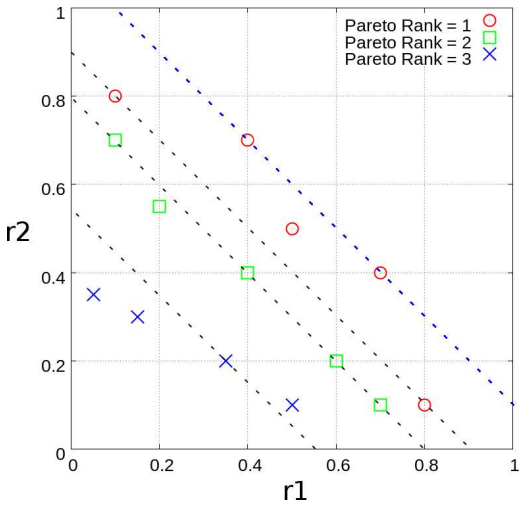
$$\begin{gathered} Maximize\ f_p(\mathbf{x}) \\ s.t.\ f_i(\mathbf{x}) \geq \epsilon_i, i = 1, \cdots, d, i \neq p \end{gathered}$$

Compared to the weighted sum method, the advantage of the ϵ-constraint method is that, by specifying ϵ parameters in each objective, this method restricts the original search space to a sub-region, which avoid redundant runs in the sense that there can be a lot of combination of weights that result in the same solution in the weighted sum method.

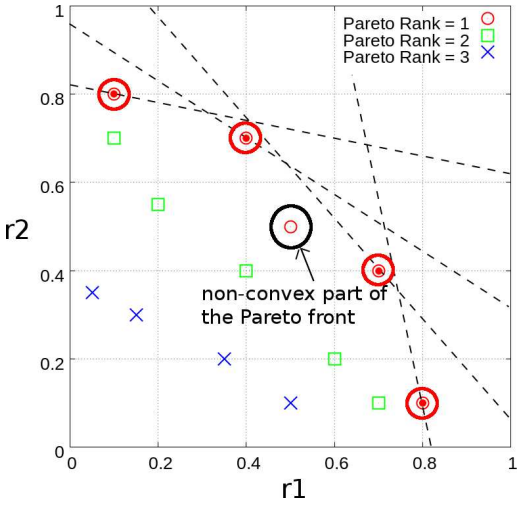
However, despite its advantage over the weighted sum method, the application of ϵ-constraint method has been prohibited due to the following drawback: the specification of constraint parameter ϵ requires strong knowledge about the objective space, which is hard to obtain, especially when the problem domain is new to algorithm designers.



(a) Preference function contours



(b) Utility function contours of the weighted sum method



(c) Optimal solutions under different weight settings

Figure 3.5: Illustration of user preference function and the weighted sum method. (a) The isolines of user preference functions are depicted as the dashed contours. Points on the same dashed curve bring the same level of satisfaction to the user. (b) The user preference function is approximated by a weighted sum ($w_1 = w_2 = 0.5$) of objective values, the isolines of which are represented as dashed lines. (c) The points that maximize the weighted sum utility function under different weight settings are marked by red circles. The point lying in the non convex region of the Pareto front (marked by the black circle) does not maximize any linear combination of objective functions.

3.2.3 Goal programming technique

Correspondingly to the ϵ-constraint method which restricts the search space to a sub-region, another choice is the goal programming technique, which indicates directly the desired region in the objective space. In goal programming, simplex method or linear programming are usually used to satisfy decision maker's goals and priorities [Charnes and Cooper, 1957; Tamiz et al., 1998].

$$Minimize\ \gamma$$
$$s.t.\ |f_i(\mathbf{x}) - f_i^*| \leq w_i\gamma, i = 1, \cdots, d, i \neq p, i = 1, \cdots, d$$

where $f^* = (f_1^*, f_2^*, \cdots, f_d^*)$ expresses the design goals.

Similarly to the ϵ-constraint, the goal programming technique requires strong knowledge about the desired region in the objective space.

3.2.4 Discussion

The common disadvantage of all preference based MOOs is that they usually require the explicit representation of user preference a priori, which is not always feasible in reality. Besides, most preference based MOO method are sensitive to the parameters ($\mathbf{w}$ in weighted sum method, ϵ in ϵ-constraint method, and f^* in goal programming).

In contrast, the ideal approaches tries to find all Pareto optimal solutions in the first place without user preference function. This is ideal in the sense that it does not require any additional information before the optimization, thus the choice of solutions is made a posteriori with the complete information about the optimal Pareto front. In practice, the determination of the whole optimal Pareto front is difficult when the objective space is continuous, and we usually search for a good approximation (e.g. a uniform sampling) of the optimal Pareto front with a limited number of points.

Although ideal approaches are expensive, they become the last resort in many real-world problems, in which the environment is unknown, and the integration of user's preference is impossible. The ideal approaches are extremely attractive in these situations as they enable the user to select a posteriori his/her preferred solution. In the following section, the state of the art in ideal approaches – the multi-objective evolutionary algorithms (MOEAs) will be presented.

3.3 Multi-Objective Evolutionary Algorithms

In this section, we firstly introduce the basics of evolutionary algorithms, which MOEAs are based on. After reviewing the main MOEA approaches, we detail the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Deb et al., 2000] algorithm, based on which many other MOEAs are developed, and one important variant of NSGA-II – the SMS-EMOA algorithm [Beume et al., 2007], which will be used in later chapters of this work.

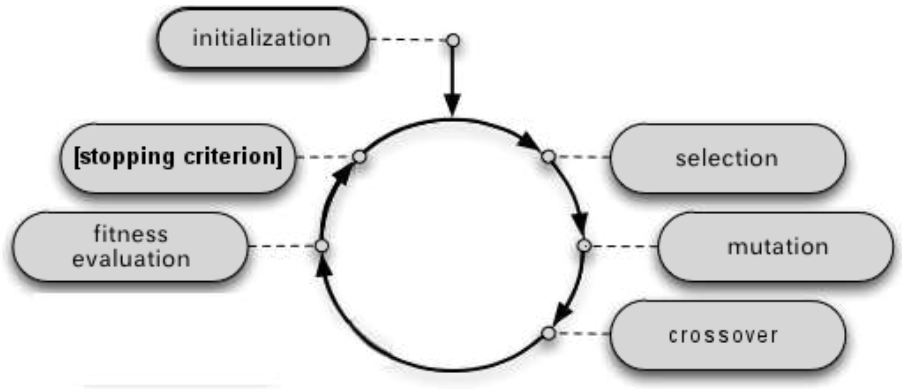


Figure 3.6: The basic evolutionary algorithm involves 4 steps: initialization, selection, variation and evaluation. Starting from a randomly or heuristically *initialized* population, all individuals in the population participate in a *selection* process which chooses better evaluated solutions to reproduce themselves. The chosen individuals (parents) go through the *variation* process (mutation and crossover) and generate off-springs. The off-spring population is then *evaluated*. According to the order defined on the evaluation results, the best individuals out of the off-spring (and optional parent) population are *selected* deterministically or stochastically and become the new generation population. The entire evolution process iterates until the stopping criterion is met.

3.3.1 Evolutionary algorithms

The term evolutionary algorithm (EA) stands for a class of stochastic optimization methods that simulate the process of natural evolution. The origins of EAs can be traced back to the late 1950s, and since 1970s several evolutionary methodologies have been proposed, including genetic algorithms [Goldberg and Holland, 1988], evolutionary programming [Fogel et al., 1966], and evolution strategies [Rechenberg, 1978].

All evolutionary algorithms define a set of candidate solutions, which are called "population" in the EA literature, and are usually randomly initialized. The population iteratively undergoes three steps : i) evaluation, ii) selection and iii) variation (Figure 3.6).

i) The evaluation corresponds to computing the value of each individual in the population, usually measured by a fitness function; ii) The selection step represents the competition for resources among individuals, and tends to select individuals which are better in the sense of fitness function; iii) The variation relies on n-nary (usually unary mutation or binary crossover) operations to generate new individuals.

Despite its simplicity, EA provides an efficient solution to the ill-posed optimization problems [Back et al., 1997]. A strength of EA is to possibly incorporate prior knowledge in the variation and in the selection step. Moreover, EAs seem to be especially suited to multi-objective optimization because they are able to capture multiple Pareto-optimal solutions in a single simulation run and may exploit similarities of solutions by recombination.

In the counterpart of the strength, the main weakness of EA is to require a huge

number of fitness functions to be evaluated [2].

3.3.2 Early MOEAs

The first way to incorporate EA into MOO is to modify the selection step. Historically, the first MOEA algorithm, Vector Evaluated Genetic Algorithm (VEGA,[Schaffer, 1985]), was a genetic algorithm with a modified selection step, where the population is divided into multiple sub-populations, and each sub-population has its own fitness-proportional based selection procedure w.r.t. one objective.

Since VEGA, the *selection step* has been recognized as the main difference between most MOEAs and their single objective analogues. An important step of understanding the selection step in MOEA was made by [Goldberg and Holland, 1988], where they discussed the concept of sorting individuals in an MOO perspective, which was later implemented in the Non-dominated Sorting Genetic Algorithm (NSGA [Srinivas and Deb, 1994]).

The common point between the early MOEAs is that they all search for a suitable selection mechanism for the MOO purpose. The main achievement of the early MOEA approaches is the creation of the non-dominated sorting procedure and various secondary sorting criteria. In the work of [Deb et al., 2000], the non-dominated sorting procedure is specified as the integer-valued fitness function – *Pareto rank* (section 3.1.2). Deb et al. [2000] suggest a fast non-dominated sort algorithm which determines the Pareto ranks of all points in P within time $\mathcal{O}(|P|^2)$ (Algorithm 3.1).

3.3.3 Quality indicators

Beside the Pareto rank indicator which measures the quality of single solutions within a single population, some quality indicators have been used by some MOEAs as the direct fitness measure of overall quality of populations (solution sets). By mapping solution sets onto $\mathbb{R}$, quality indicators provide a total order among sets. Then, even for incomparable non-dominated sets, one can say which one is better w.r.t. a given quality indicator.

Basically, there are two types of quality indicators – unary and binary indicators. In the first case, the *unary quality indicator* $I : \mathbb{P}(\mathbb{R}^d) \to \mathbb{R}$ maps populations onto $\mathbb{R}$, therefore introducing a total order in the power set of the decision space $\mathcal{X}$ [3]. However, unary quality indicators are limited in the sense that there is no intrinsic total order among point sets, therefore no unary quality indicator is able to judge whether one non-dominated set P_1 is better than another non-dominated set P_2 ($P_1 \rhd P_2$) [Zitzler et al., 2003].

A unary quality indicator is said to be Pareto compliant if $I(P_1) > I(P_2)$ for all pairs of point sets P_1, P_2 such that P_1 is better than P_2 (Definition 9).

$$P_1 \rhd P_2 \Rightarrow I(P_1) > I(P_2)$$

[2] Some learning mechanisms incorporating the approximation of fitness function evaluation have been developed to address this limitation (see for instance [Loshchilov, 2013]), but this is beyond the scope of this manuscript.

[3] $\mathbb{P}(\mathbb{R}^d)$ is the power set on $\mathbb{R}^d$.

Algorithm 3.1: Fast non-dominated sort algorithm

Input: Point set P
Output: A list of non-dominated sets of different Pareto ranks $\mathcal{F}_i, i \in \mathbf{N}$
for $p \in P$ **do**
  $n_p \leftarrow 0$ //n_p notes the number of points that dominate p
  for $q \in P$ **do**
    if $p \succ q$ **then**
      $S_p \leftarrow S_p \cup \{q\}$ //S_p stores the points that are dominated by p
    else if $q \succ p$ **then**
      $n_p \leftarrow n_p + 1$
    end if
  end for
  if $n_p = 0$ **then**
    //The points that are not dominated by any other points are
    //included in the first non-dominated set $\mathcal{F}_1$.
    $\mathcal{F}_1 \leftarrow \mathcal{F}_1 \cup \{p\}$
  end if
end for
$i \leftarrow 1$
while $\mathcal{F}_i \neq \emptyset$ **do**
  $\mathcal{H} = \emptyset$ //$\mathcal{H}$ stores the remaining non-dominated points in the reduced point set P
  for $p \in \mathcal{F}_i$ **do**
    for $q \in S_p$ **do**
      $n_q \leftarrow n_q - 1$
      if $n_q = 0$ **then**
        //new non-dominated sets are constructed in a similar way to $\mathcal{F}_1$
        $\mathcal{H} = \mathcal{H} \cup \{q\}$
      end if
    end for
  end for
  $i \leftarrow i + 1$
  $\mathcal{F}_i = \mathcal{H}$
end while
return $\{\mathcal{F}_k | k = 1, 2, \cdots, i-1\}$

More sophisticated binary quality indicators are defined on pairs of solution sets $I : \mathbb{P}(\mathbb{R}^d) \times \mathbb{P}(\mathbb{R}^d) \to \mathbb{R}$. Binary quality indicators map pairs of solution sets onto $\mathbb{R}$. A reasonable requirement for the binary quality indicator is to preserve the Pareto order among solution sets. A trivial example of such indicator is

$$I(P_1, P_2) = \begin{cases} 1 & \text{if } P_1^* \rhd P_2^* \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

3.3.4 Unary quality indicators

In this section, some of the most popular unary quality indicators are presented.

Generational distance

Let P^{**} denote the true Pareto front, and let P^* denote the empirical Pareto front obtained by a search process. The generational distance (GD) measures the average distance from points in P^* to the true Pareto front P^{**} [Van Veldhuizen, 1999].

$$GD(P^*) = \frac{\sqrt{\sum_{i=1}^{|P^*|} d_i^2}}{|P^*|}$$

where d_i is the Euclidean distance between the i-th point in P^* and its nearest neighbour in P^{**}.

Inverted generational distance

The inverted generational distance (IGD) is defined as the average distance of points in P^{**} to their nearest neighbour in P^* [Van Veldhuizen, 1999].

$$IGD(P^*) = \frac{\sqrt{\sum_{i=1}^{|P^{**}|} {d_i^*}^2}}{|P^{**}|}$$

where d_i^* is the Euclidean distance between the i-th point in P^{**} and its nearest neighbour in P^*.

GD measures the average closeness of points in P^* to the true Pareto front, while IGD indicates whether points in P^{**} are all uniformly approximated by P^*. In order to best approximate the Pareto front P^{**}, both generational and inverted generational distances should be minimized. Although GD and IGD are used in numerous MOO problems as quality indicators of solution sets since Van Veldhuizen [1999], they are proven not to be Pareto-compliant by Zitzler et al. [2003]. An example of this is shown by Figure 3.7, in which a solution set corresponding to a smaller GD is dominated by a solution set with higher GD.

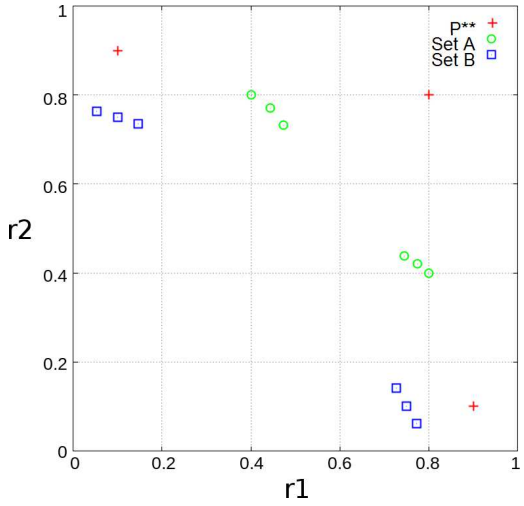


Figure 3.7: With respect to the optimal Pareto front (red stars), the set B (blue squares) corresponding to a smaller generational distance is dominated by set A (green circles) with a greater generational distance.

Hypervolume Indicator

The hypervolume indicator [Zitzler and Thiele, 1998] defines a scalar measure of solution sets in the multi-objective space as follows.

Definition 11. *Given $P \subseteq \mathbb{R}^d$ a set of objective vectors, and the reference point $z \in \mathbb{R}^d$ such that it is dominated by every $p \in P$, the hypervolume indicator (HV) of P is the measure of the set of points dominated by some point in P and dominating z:*

$$HV(P; z) = \mu(\{x \in \mathbb{R}^d : \exists p \in P \ s.t. \ p \succeq x \succeq z\})$$

where μ is the Lebesgue measure on $\mathbb{R}^d$ (Figure 3.3.4).

It is clear that all dominated points in P can be removed without modifying the hypervolume indicator ($HV(P; z) = HV(P^*; z)$). As shown by [Fleischer, 2003], the hypervolume indicator is maximized iff points in P^* belong to the Pareto front of the MOO problem. [Auger et al., 2009] show that, for $d = 2$, for a number K of points, the hypervolume indicator maps a multi-objective optimization problem defined on $\mathbb{R}^d$, onto a single-objective optimization problem on $\mathbb{R}^{d \times K}$, in the sense that there exists at least one set of K points in $\mathbb{R}^d$ that maximizes the hypervolume indicator w.r.t. z.

3.3.5 Modern MOEAs

The history of early MOEA approaches mainly is the search for suitable selection criterion in the multi-objective setting. While the modern MOEAs are characterized for their use of quality indicators as the secondary selection criterion. The Indicator Based Evolution Algorithm (IBEA [Zitzler and Künzli, 2004]) is one of the first algorithms that use the

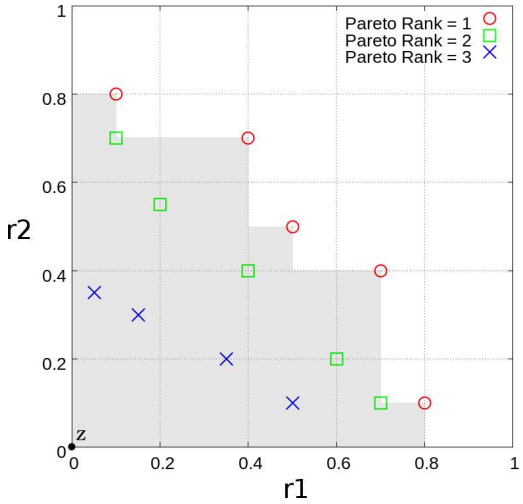


Figure 3.8: The hypervolume indicator of the point set w.r.t. reference point z in the lower-left corner is the surface of the *shaded region*.

quality indicators as selection criterion in the evolution process. Afterwards, the SMS-EMOA [Beume et al., 2007] implemented the hypervolume indicator as the second sorting criterion in the non-dominated sorting algorithm. This allows extending the NSGA-II algorithm towards the many-objective optimization problem where the objective number $d > 3$ (more in section 3.3.7).

In the following sections, NSGA-II and SMS-EMOA are respectively introduced as representatives of early and modern MOEA approaches.

3.3.6 NSGA-II algorithm

Despite the diversity of early MOEAs, we consider the NSGA-II [Deb et al., 2000] as a representative of them because of the following facts: i) NSGA-II is the best known MOO algorithm which is frequently used as the baseline algorithm in the MOO literature; ii) it serves as a basis on which many new MOEAs are developed (usually by modifying the secondary sorting criterion, such as SMS-EMOA [Beume et al., 2007]); iii) it has been successfully implemented in many multi-objective sequential decision making problems, including the grid scheduling problem [Yu et al., 2008] which is used as a benchmark in the following of this work (Chapter 6).

The NSGA-II is an improved version of NSGA [Srinivas and Deb, 1994], in which the concept of non-dominated sorting was first defined. In NSGA-II, the order of individuals in the selection process of evolutionary algorithm is determined by two indicators:

> Firstly, the dominance-based quality indicator i_{rank} is defined by the Pareto rank criterion: the fast non-dominated sorting algorithm (Algorithm 3.1) partitions the objective vectors of the current population into several layers according to their Pareto ranks, and each layer of vectors compose a non-dominated set (Figure .3.1).

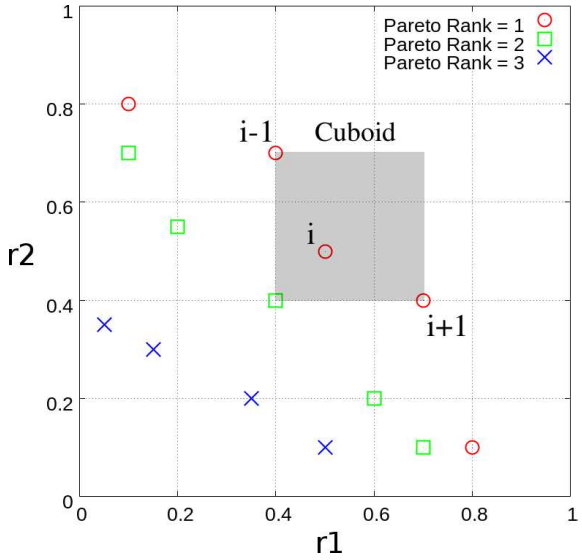


Figure 3.9: Calculation of the crowding distance.

Secondly, in order to distinguish vectors of the same Pareto rank, the second quality indicator $i_{distance}$, defined by *crowding distance* is applied (Figure 3.9). The crowding distance measures the population density around a vector $\mathbf{y}$ within a non-dominated set P. It is computed as follows:

$$i_{distance}(\mathbf{y}, P) = \sum_{i=1}^{d} \frac{f_i(\mathbf{y}_i^+) - f_i(\mathbf{y}_i^-)}{\max(f_i(\mathbf{y}), \mathbf{y} \in P) - \min(f_i(\mathbf{y}), \mathbf{y} \in P)} \tag{3.5}$$

where $\mathbf{y}_i^+$ and $\mathbf{y}_i^-$ are respectively the right and left neighbors of $\mathbf{y}$ on the i-th objective.

The two indicators together define a *crowded comparison operator* ($\geq_{ns}$) as:

$$\begin{aligned} \mathbf{y} \geq_{ns} \mathbf{y}' \text{ if } \quad & i_{rank}(\mathbf{y}, P) < i_{rank}(\mathbf{y}', P) \\ & \text{or} \\ & i_{rank}(\mathbf{y}, P) = i_{rank}(\mathbf{y}', P) \text{ and } i_{distance}(\mathbf{y}, P) > i_{distance}(\mathbf{y}', P) \end{aligned} \tag{3.6}$$

Formally, the NSGA-II algorithm is described in Algorithm 3.2. In iteration $t = 0$, a population $Q^{t=0}$ of μ individuals is randomly initialized. In each iteration t, the population Q^t generates λ offsprings using two variation operators – crossover $\mathcal{C}$ (with probability p_c) and mutation $\mathcal{M}$ (with mutation p_m). Each newly generated solution $\mathbf{x}'_k$ is evaluated and its objective vector $f(\mathbf{x}'_k)$ will later participate in the selection process. After the procedure of variation, $\mu + \lambda$ parent and offspring individuals are sorted according to the total order relation $\geq_{ns}$ defined above.

The main advantage of NSGA-II is that the crowding distance gives a measure of population density without requiring a user-defined parameter. However, in higher dimensional

Algorithm 3.2: $(\mu + \lambda)$-NGSA-II algorithm

Input: parent population size μ, offspring population size λ, crossover rate p_c, mutation rate p_m, crossover operator $\mathcal{C}$, mutation operator $\mathcal{M}$, tournament size t_{tour}
Initialize: $t \leftarrow 0$, build a (usually uniform) parent population $Q^{t=0}$
repeat
  for $k = 1, \cdots, \lambda$ **do**
    //Choose two parents according to the tournament-based selection rule
    repeat
      $i_1 \leftarrow TournamentSelection(|Q^t|, t_{tour})$
      $i_2 \leftarrow TournamentSelection(|Q^t|, t_{tour})$
    until $i_1 \neq i_2$
    if $\mathbb{U}(0, 1.0) \leq p_c$ **then**
      $\{o_1, o_2\} \leftarrow \mathcal{C}(\mathbf{x}_{i_1}, \mathbf{x}_{i_2})$
      //crossover operation with probability p_c between the selected parents
    else
      $o_1 \leftarrow \mathbf{x}_{i_1}, o_2 \leftarrow \mathbf{x}_{i_2}$
    end if
    $\mathbf{x}'_k \leftarrow o_1$ or o_2 (each with probability 0.5)
    if $\mathbb{U}(0, 1.0) \leq p_m$ **then**
      $\mathbf{x}'_k \leftarrow \mathcal{M}(\mathbf{x}'_k)$
      //mutation operation with probability p_m over the generated offspring
    end if
    $Q^t \leftarrow Q^t \cup \{f(\mathbf{x}'_k)\}$ //evaluation operation
  end for
  $Sort(Q^t, \geq_{ns})$
  //sort the Q^t point set according to the crowded comparison operator
  $Q^{t+1} \leftarrow Q^t[1 \cdots \mu]$ //select the first μ elements in Q^t
  $t \leftarrow t + 1$
until stopping criterion is met

Algorithm 3.3: TournamentSelection function

Input: population size S, tournament size t_{tour}
Output: index of the tournament winner
$best \leftarrow \mathbb{U}(1, S)$ //uniformly choose a number between 1 and S
for $k = 2, \cdots, t_{tour}$ **do**
  $compete \leftarrow \mathbb{U}(1, S)$ //randomly choose another number between 1 and S
  if $f(\mathbf{x}_{compete}) \succ f(\mathbf{x}_{best})$ **then**
    $best \leftarrow compete$
  end if
end for
return *best*

objective spaces ($d > 3$), the crowding distance may not properly estimate the population density, because points in high dimensional space are far from each other, according to the famed curse of curse of dimensionality [Bellman, 1957].

3.3.7 SMS-EMOA algorithm

In order to address the drawback of the crowded distance indicator in NSGA-II, the S-metric selection evolutionary multi-objective optimization algorithm (SMS-EMOA) [Beume et al., 2007] has been proposed, which replaces the second quality indicator in NSGA-II by the hypervolume indicator. Note the candidate solution set (population) by P. For any individual $\mathbf{y} \in P$, its hypervolume contribution is defined as:

$$i_{hypervolume}(\mathbf{y}, P) = HV(P; z) - HV(P/\{\mathbf{y}\}; z) \tag{3.7}$$

The dominance-based quality indicator i_{rank} and hypervolume contribution indicator $i_{hypervolume}$ together define a hypervolume contribution operator ($\geq_{hv}$) as:

$$\begin{aligned} \mathbf{y} \geq_{hv} \mathbf{y}' \text{ if } \quad & i_{rank}(\mathbf{y}, P) > i_{rank}(\mathbf{y}', P) \\ & \text{or} \\ & i_{rank}(\mathbf{y}, P) = i_{rank}(\mathbf{y}', P) \text{ and } i_{hypervolume}(\mathbf{y}, P) > i_{hypervolume}(\mathbf{y}', P) \end{aligned} \tag{3.8}$$

Then, SMS-EMOA is carried out in the same way as NSGA-II, only with a modified selection step: the $Sort(Q^t, \geq_{ns})$ sorting procedure is modified to $Sort(Q^t, \geq_{hv})$.

The main disadvantage of SMS-EMOA is in the computational complexity of the hypervolume indicator, which increases exponentially w.r.t. the number of objectives d (the complexity $\mathcal{O}(|P|^{d/2})$ for $d > 3$ [Beume et al., 2009]). To make the hypervolume indicator based algorithm computationally feasible for large d, Monte Carlo simulation schemes have been proposed to approximate the exact hypervolume values [Bader and Zitzler, 2011].

3.4 Discussion

MOO techniques have been deployed with success in a wide variety of real world problems, including planning [Saadatseresht et al., 2009; Yu et al., 2008], circuit design [Palmers et al., 2009; Zhao and Jiao, 2006], robotics [Saravanan et al., 2009], bioinformatics [Shin et al., 2005], image processing [Lazzerini et al., 2010] and traffic engineering [Uhlig, 2005]. The main open problems in MOO can be sketched as follows.

- Noisy multi-objective optimization: Noisy optimization is a big issue both theoretically and computationally. It is particularly important in MOO as noise can perturb Pareto dominance relations, all the more so as the number of objectives increases. To develop efficient algorithms, a model of the noise should preferably be identified to propose adapted heuristics to cope with it. Although some work has been done on this [Deb and Gupta, 2006; Goh and Tan, 2007; Heidrich-Meisner and Igel, 2009], the fundamental issues of noise modelling [Beyer and Sendhoff, 2007; Finck et al., 2010] have not much been touched in MOO to our best knowledge.

- Many objectives: The complexity of handling numerous objectives has attracted growing attention [Purshouse and Fleming, 2007; Ishibuchi et al., 2008; Zou et al., 2008; Bader and Zitzler, 2011]. Due to the curse of dimensionality, a large number ($d \geq 3$) of objectives introduce extra difficulties w.r.t. computation, visualization, and decision making for the MOO. Ishibuchi et al. [2008] show that many algorithms that perform well for a few objectives scale poorly in the number of objectives, necessitating special algorithms for the many-objective setting. As already mentioned, the state of the art MOEAs such as NSGA-II are not effective in solving optimization problems with more than three objectives.

- Description of approximated Pareto front: The purpose of ideal MOO method is to approximate the set of Pareto optimal solutions P^{**}. Although most current MOOs use a finite number of solutions (points) to approximate the optimal Pareto front, it would be interesting to investigate how to describe an approximated Pareto front in other ways. Some steps have been made in this direction. In the continuous case, one may consider first-order or higher-order approximations [Loshchilov, 2013].

Chapter 4

Multi-Objective Reinforcement Learning

Multi-Objective Reinforcement Learning (MORL) is an extension of reinforcement learning to the multi-objective setting. This chapter firstly introduces the formal background of MORL pioneered by Gábor et al. [1998]. Then different categories of MORL algorithms are presented. MORL applications are discussed in the end of this chapter.

4.1 MORL background

Multi-Objective Reinforcement Learning (MORL), aimed at multi-objective sequential decision making, addresses problems formulated as multi-objective Markov decision process (MOMDP). This section firstly presents the MOMDP framework [Roijers et al., 2013], then the problems raised in the MOMDP are discussed.

4.1.1 MOMDP

The formal setting in multi-objective Markov decision process (MOMDP) is the same as in MDP (section 2.1.1), which consists of a quadruplet $(\mathcal{S}, \mathcal{A}, p, \mathbf{r})$ involving the state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition probability function p, and the reward function $\mathbf{r}$. With respect to MDP, the only modification of MOMDP is that the reward function $\mathbf{r} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ describes a d-dimensional vectorial reward, (e.g. cost, risk, robustness) instead of a single scalar value (quality).

Similarly to Chapter 2, we note $\mathbf{r}(s_t, a_t) = (r_1(s_t, a_t), r_2(s_t, a_t), \cdots, r_d(s_t, a_t))$ as the vector of rewards received at time t, the value functions $\mathbf{V}^\pi : \mathcal{S} \to \mathbb{R}^d$ and $\mathbf{Q}^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ specify the expected return of each objective as follows:

$$\begin{aligned} \mathbf{V}^\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot \mathbf{r}(s_t, a_t) | s_0 = s, a_t = \pi(s_t)\right] \\ &= (V_1^\pi(s), V_2^\pi(s), \cdots, V_d^\pi(s)) \end{aligned} \tag{4.1}$$

$$\begin{aligned} \mathbf{Q}^\pi(s, a) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot \mathbf{r}(s_t, a_t) | s_0 = s, a_0 = a, a_t = \pi(s_t) \text{ for } t > 1\right] \\ &= (Q_1^\pi(s, a), Q_2^\pi(s, a), \cdots, Q_d^\pi(s, a)) \end{aligned} \tag{4.2}$$

For an MOMDP, given a set of policies $\Pi = \{\pi_1, \pi_2, \cdots, \pi_k\}$ and a state s, each policy $\pi \in \Pi$ corresponds to a value vector $\mathbf{V}^\pi(s)$. Note the set of value vectors for Π under s by $P_{s;\Pi} = \{\mathbf{V}^\pi(s), \pi \in \Pi\}$. Then we say that the policy $\pi \in \Pi$ belongs to the Pareto front of Π under state s iff $\mathbf{V}^\pi(s) \in P^*_{s;\Pi}$.

4.1.2 Multi-objective generative model

Like in the MDP case, the MOMDP is sometimes replaced by a generative model $\mathcal{M}_d : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathbb{R}^d$, which maps the state-action pair (s, a) to the next state s' and the associated vectorial reward $\mathbf{r}$.

4.1.3 MOMDP difficulties

As discussed in Chapter 3, the key impact of the multi-objective setting is that there is no longer a total order relation on rewards $\mathbf{r}$ and value functions $\mathbf{V}, \mathbf{Q}$. Therefore the standard MDP solutions do not work directly in MOMDP. Two main approaches have been proposed in the literature. The first one is based on the single-policy method, which aggregates multiple objectives into a single one through the use of scalarization function (section 4.2). At this point, we will further distinguish between the linear scalarization function and the non-linear scalarization function. The second one is based on the multiple-policy method, which aims at finding all policies with Pareto optimal values in the multi-dimensional space.

4.2 Scalarization functions

Definition 12. *A scalarization function $g_\rho : \mathbb{R}^d \rightarrow \mathbb{R}$, is a function that maps the vectorial reward $\mathbf{r}(s, \pi(s)) = (r_1(s, \pi(s)), r_2(s, \pi(s)), \cdots, r_d(s, \pi(s)))$ onto a scalar value.*

$$r_\rho(s, \pi(s)) = g_\rho(\mathbf{r}(s, \pi(s))) \tag{4.3}$$

where $\rho = (\rho_1, \rho_2, \cdots, \rho_l)$ is a parameter vector of g.

Beside the vectorial rewards, g_ρ is applicable to the value function $\mathbf{V}^\pi(s)$ as well. We note

$$V^\pi_\rho(s) = g_\rho(\mathbf{V}^\pi(s)) \tag{4.4}$$

Definition 13. *A scalarization function $g_\rho : \mathbb{R}^d \rightarrow \mathbb{R}$ is Pareto-compliant if the Pareto-dominance relation between vectorial rewards $\mathbf{r} = (r_1, r_2, \cdots, r_d)$ and $\mathbf{r}' = (r'_1, r'_2, \cdots, r'_d)$ is respected :*

$$\forall i, r_i \geq r'_i \wedge \exists j, r_j > r'_j \Rightarrow g_\rho(\mathbf{r}) > g_\rho(\mathbf{r}') \tag{4.5}$$

The Pareto-compliant scalarization only requires that, all other things being equal, getting more reward for one objective is always better, being reminded that all considered objectives are to be maximized. To the best of our knowledge, all scalarization functions in the MORL literature are Pareto-compliant.

Three classes of scalarization functions are distinguished in the MORL literature and are presented in the following sections. The first class is the linear scalarization function (section 4.2.1). The second class is the non-linear scalarization function (section 4.2.2). The third class, which is recently studied in both MOEA and MORL communities, is

the population-based scalarization (section 4.2.3, [Ishibuchi et al., 2008; Wang and Sebag, 2012; Van Moffaert et al., 2013]). In the first and the second case, we have the parameter ρ fixed, while in the third case, the scalarization function is basically calculated from the current archive P of previously found solutions, which is updated by the MORL algorithm.

Except for the linear-scalarization class, standard MDP approaches are not directly applicable to other scalarized MOMDP scalarized in other ways.

4.2.1 Linear scalarization functions

A linear scalarization function computes the inner product of a vectorial reward $\mathbf{r} = (r_1, r_2, \cdots, r_d)$ and a weight parameter $\mathbf{w} = (w_1, w_2, \cdots, w_d)$:

$$g_{\mathbf{w}}(\mathbf{r}) = \mathbf{r} \cdot \mathbf{w} = \sum_{i=1}^{d} r_i \cdot w_i \tag{4.6}$$

Each element of $\mathbf{w}$ specifies the relative importance of each objective. With no loss of generality, all elements in $\mathbf{w}$ are non-negative and sum to 1.

The main merit of the linear-scalarization function is that it preserves the Bellman equation :

$$\begin{aligned}
g_{\mathbf{w}}(\mathbf{V}^{\pi}(s)) &= \sum_{i=1}^{d} w_i \cdot V_i^{\pi}(s) \\
&= \sum_{i=1}^{d} w_i \cdot \left(r_i(s, \pi(s)) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') V_i^{\pi}(s) \right) \\
&\quad \text{(the Bellman equation holds in each objective)} \\
&= \sum_{i=1}^{d} r_i(s, \pi(s)) \cdot w_i + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') \sum_{i=1}^{d} V_i^{\pi}(s) \cdot w_i \\
&= g_{\mathbf{w}}(\mathbf{r}(s, \pi(s))) + \gamma \cdot \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') g_{\mathbf{w}}(\mathbf{V}^{\pi}(s))
\end{aligned} \tag{4.7}$$

The additivity of state values allows the implementation of Bellman operator, and the linear-scalarized optimal value of states in MOMDP can be calculated by the standard MDP approaches.

However, the price to pay for this simplicity is that the linear scalarization function does not allow the discovery of Pareto optimal solutions lying in the non-convex regions of the Pareto front.

4.2.2 Non-linear scalarization functions

Non-linear scalarization function $g_{\rho} : \mathbb{R}^d \to \mathbb{R}$ maps the vectorial reward onto $\mathbb{R}$ in a non-linear way.

The well-known Tchebycheff function is an example of this class:

$$g_{\mathbf{m},\mathbf{w}}(\mathbf{r}) = -||\mathbf{w} \cdot (\mathbf{m} - \mathbf{r})||_\infty \tag{4.8}$$

where $\mathbf{m} = (m_1, m_2, \cdots, m_d)$ is an Utopian/ideal point, representing the desired values in each objective, and $\mathbf{w}$ are weights assigned to each objective. Note that any Pareto optimal solution can be reached by maximizing the Tchebycheff scalarization function with a proper choice of $\mathbf{w}$, and any Tchebycheff-optimal solution is Pareto-optimal [Bowman Jr, 1976]. The main difficulty for implementing the Tchebycheff function is on the choice of $\mathbf{m}$ and $\mathbf{w}$.

The cumulative property of rewards does not hold for non-linearly scalarized rewards. Take the Tchebycheff scalarization function for example. If $\mathbf{m} = (1, 1)$, and under a policy π, we have $\mathbf{r}_0 = \mathbf{r}(s_0, a_0) = (0, 1), \mathbf{r}_1 = \mathbf{r}(s_1, a_1) = (1, 0), \mathbf{r}_t = \mathbf{r}(s_t, a_t) = (0, 0)$ for $t > 1$, $\mathbf{w} = (0.3, 0.7)$ and $\gamma = 1$. We have $\mathbf{V}^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot \mathbf{r}_t\right] = (1, 1)$. Then $g_{\mathbf{m},\mathbf{w}}(\mathbf{r}_0) = -0.3, g_{\mathbf{m},\mathbf{w}}(\mathbf{r}_1) = -0.7$ and $g_{\mathbf{m},\mathbf{w}}(\mathbf{r}_t) = -0.7$ for $t > 1$, but $g_{\mathbf{m},\mathbf{w}}(\mathbf{V}^\pi) = 0 \neq \sum_{t=0}^{\infty} \gamma^t \cdot g_{\mathbf{m},\mathbf{w}}(\mathbf{r}_t)$.

Therefore, standard RL methods which exploit the additivity of expected cumulative reward are no longer applicable when the scalarization function is non-linear.

4.2.3 Population-based scalarization functions

Population-based scalarization functions maps a vectorial reward to a scalar value based on some archive P of the vectorial rewards discovered previously. One example function of this class is the hypervolume indicator introduced in section 3.3.3. Such scalarization functions are extensively used in MOEAs in the form of quality indicators (section 3.3.3). They attract increasingly attention from the MORL community ([Wang and Sebag, 2012; Van Moffaert et al., 2013]) as well.

As already mentioned, population-based scalarization approaches have the possibility to search for solution set on the Pareto front in a single trial. As population-based scalarization functions do not belong to the linear scalarization class, the additive assumption (Eq.(4.7)) does not hold. Besides, the dynamic nature of the population-based scalarization function (due to the evolution of P) makes it difficult to compare the value of solutions along time.

Following the approach of [Vamplew et al., 2010], the MORLs can be divided into two classes based on the number of policies that they search for. One class aims to learn the single policy that best satisfies the preferences between objectives which are specified by the user or derived from the problem domain. We refer to these as *single-policy algorithms*. The second class searches for a set of policies which lie on or are close to the Pareto front. We refer to this class as *multiple-policy algorithms*. The single and multiple-policy approaches in MORL respectively upgrade the preference based and ideal approaches in the MOO literature. The difference is that the MORL proceeds in the MDP framework.

In the following, the single- and multiple-policy MORL algorithms are presented respectively in section 4.3 and section 4.4.

4.3 Single-policy MORLs

The majority of MORL algorithms proposed so far are of the single-policy nature as they learn a single policy [Gábor et al., 1998; Castelletti et al., 2002; Mannor and Shimkin, 2004; Natarajan and Tadepalli, 2005; Tesauro et al., 2007]. The main difference between existing single-policy algorithms regards the scalarization function used to express the user's preferences.

4.3.1 Linear scalarization based single-policy algorithms

The simplest way to express the user preference is to use the linear-scalarization function (section 4.2.1), which has been employed by [Castelletti et al., 2002; Natarajan and Tadepalli, 2005; Tesauro et al., 2007]. This approach requires little knowledge about solution distributions in the objective space, and the relative importance of objectives is represented by the weight parameter $\mathbf{w}$. As the linear-scalarization keeps the cumulative property of rewards in each objective, standard RL methods can be applied directly on the scalarized values functions.

The critical issue in linear-scalarization-based single-policy MORL is how to determine the weights which best describe the user preferences. This problem becomes even more challenging when the magnitude of rewards in each objective is unknown.

4.3.2 Non-linear scalarization based single-policy algorithms

Gábor et al. [1998] provide the earliest example of a non-linear scalarization-based single-policy MORL – the thresholded lexicographic Q-learning (TLQ-learning). This algorithm is designed for problems with constraints in some of the objectives. The preference over policies is defined using 1) an order on the objectives; 2) constraints ($f_i \geq C_i$) on objectives $i = 1, 2, \cdots, d-1$. Specifically, the order between two value vectors $\mathbf{Q}$ and $\mathbf{Q'}$ is determined by Algorithm 4.1. The first, out of $\mathbf{Q}$ and $\mathbf{Q'}$, to violate one of the ordered constraints, is said to be dominated by the second. Otherwise (both $\mathbf{Q}$ and $\mathbf{Q'}$ satisfy all constraints), the best solution is the one with better value w.r.t. the last objective f_d.

Through the use of thresholded lexicographic ordering on the objectives, a total order is recovered among vectors in the objective space. Then the standard Q-learning technique can be applied to maximize the resulting scalarized values $\prod_i^{d-1} \mathbf{1}_{Q_i \geq C_i} \cdot Q_d$. The convergence of TLQ-learning is proven by the authors [Gábor et al., 1998].

Mannor and Shimkin [2004] also use preferences defined on the objective space to specify the desired characteristics of the policy being learnt. In this case a target region in the objective space is defined in which expected return of the policy should fall.

[Perny and Weng, 2010] proposed another single-policy MORL solution with non-linear scalarization function, which implements a linear programming method to solve the MOMDP scalarized using the Tchebycheff function (section 4.2), which requires the definition of an Utopian point and a weight setting to represent the desired optimization direction in the objective space.

Algorithm 4.1: TLQ-learning

Input: the threshold value (minimum acceptable value) C_i for objective $i = 1, 2, \cdots, d$ (for the unconstrained objective, $C_d = +\infty$), vector $\mathbf{Q} = (Q_1, \cdots, Q_d)$, vector $\mathbf{Q}' = (Q'_1, \cdots, Q'_d)$
Output: boolean – whether $\mathbf{Q}$ is better than $\mathbf{Q}'$
for all $i = 1, 2, \cdots, d-1$ **do**
  if $\mathbf{Q}_i < C_i$ and $Q'_i \geq C_i$ **then**
    return false
  else if $\mathbf{Q}_i \geq C_i$ and $Q'_i < C_i$ **then**
    return true
  end if
end for
if $Q_d > Q'_d$ **then**
  return true
else
  return false
end if

In summary, existing non-linear scalarization based MORLs usually require an explicit expression of the optimization targets, which makes the design of scalarization functions more intuitive.

4.4 Multiple-policy MORLs

Multiple-policy algorithms aim to learn a set of policies that lie on the Pareto front. By considering diverse parameters ρ in the scalarization function (Eq.(4.3)), different Pareto optimal policies can be discovered.

Compared to single-policy algorithms, multiple-policy algorithms share the same advantages as the ideal MOO approaches : no prior knowledge about the user preferences is required and the set of solution policies built by the algorithm allow the user to select a policy based on his or her private criteria and the trade-off among the objectives empirically disclosed by the Pareto front.

Based on the nature of scalarization functions, multiple-policy MORLs are categorized into three classes : linear scalarization based, non-linear scalarization based, and population-based multiple-policy algorithms, which are presented in the following sections.

4.4.1 Linear scalarization based multiple-policy algorithms

Most existing multiple-policy algorithms are based on the linear scalarization function. Examples of this class include [Natarajan and Tadepalli, 2005; Chatterjee, 2007; Barrett and Narayanan, 2008; Lizotte et al., 2012]. By providing multiple weight settings to the linear scalarization function (section 4.2.1), these algorithms use standard RL methods to

produce multiple policies in repetitive runs.

In this section, we introduce the Multi-Objective Q-learning (MOQ-learning, [Vamplew et al., 2010]) algorithm as a representative of all other linear scalarization based multiple-policy MORLs in the sense that it yields all policies found by other linear-scalarisation based approaches, provided that a sufficient number of weight settings be considered.

Formally, MOQ-learning optimizes independently m scalar RL problems through Q-learning, where the j-th problem considers reward $R_j = \sum_{i=1}^{d} w_{ji} \cdot r_i,\ w_{ji} \geq 0, j = 1, 2, \cdots, m$ define the m weight settings of MOQ-learning, $\sum_{i=1}^{d} w_{ji} = 1$, and r_i represents the reward in the i-th objective, $i = 1, 2, \cdots, d$. The computational effort allocated to each weight setting is further equally divided into n_{tr} training phases; after the k-th training phase, the performance of the j-th weight setting is measured by the d-dimensional vectorial reward, noted $R_{j,k}$, of the current greedy policy. The m vectorial rewards of all weight settings $\{R_{1,k}, R_{2,k}, \ldots, R_{m,k}\}$ together compose the Pareto front of MOQ-learning at training phase k.

However, in the simplest case, if there are only l values evenly distributed over $[0, 1]$ $\{0, \frac{1}{l-1}, \frac{2}{l-1}, \cdots, \frac{l-1}{l-1}\}$ allowed to be taken as weights w_i for each objective $i = 1, 2, \cdots, d$, and $\sum_{i=1}^{d} w_i = 1$, there are in total $m = \begin{pmatrix} l \\ d-1 \end{pmatrix}$ possible combinations of weight settings to be taken. If the standard RL technique is applied on each weight setting, and the computational effort is equally divided between the resulting m scalar RL problems, then the complexity of MOQ-learning algorithm will increase exponentially w.r.t. the number d of objectives.

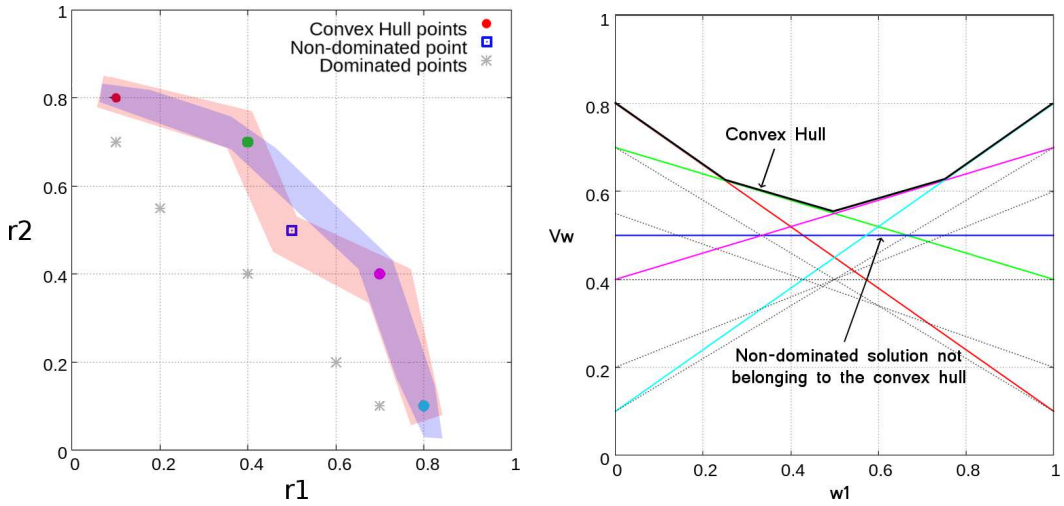
Several other multiple-policy MORL algorithms have been proposed to handle the complexity problem of linear-scalarization based multiple policy MORL [Natarajan and Tadepalli, 2005; Tesauro et al., 2007; Barrett and Narayanan, 2008; Lizotte et al., 2012]. The differences between the above algorithms are how they share the information between different weight settings and which weight settings they choose to optimize. Natarajan and Tadepalli [2005] show that the efficiency of linear-scalarization based MORL can be improved by sharing information between different weight settings.

In the case where the Pareto front is known, the design of the weight settings is made easier provided that the Pareto front is convex. When the Pareto front is unknown, the alternative investigated by Barrett and Narayanan [2008] is to maintain Q-vectors instead of Q-values for each pair (state, action). A subset of all Q-vectors discovered in the objective space, referred to as convex hull, is defined as follows :

Definition 14. *For an MOMDP, given a set of policies* $\Pi = \{\pi_1, \pi_2, \cdots, \pi_k\}$ *and a state* s*, the policy* $\pi \in \Pi$ *belongs to the convex hull (CH) of* Π *under state* s *if there exist a* $\mathbf{w}$ *that maximizes the linearly scalarized value function:*

$$CH(\Pi, s) = \{\pi | \pi \in \Pi, \exists \mathbf{w} \in \mathbb{R}^d \ s.t.\ \forall \pi' \in \Pi, \mathbf{w} \cdot \mathbf{V}^{\pi}(s) \geq \mathbf{w} \cdot \mathbf{V}^{\pi'}(s)\} \tag{4.9}$$

Figure 4.1 illustrates the concept of a convex hull of a set of policies : the convex hull on the solution set is marked by the blue shadow in Figure 4.1(a). The dual representation of these points on the scalarized value space is shown by bold lines in Figure 4.1(b), which



(a) Points in the objective space

(b) Linear scalarized values of points

Figure 4.1: The convex hull of points in the objective space. Each point in (a) represents the bi-objective value of a given policy and each line in (b) shows the dual representation of these points in the space of linearly scalarized policy values with the x-axis representing the weight w_1 for objective 1 (then $w_2 = 1 - w_1$), and the y-axis representing the scalarized value of the policies. The convex hull is shown as circles in (a), and solid lines in (b). The Pareto front consists of all circles and the blue square in (a), among which the non-dominated points are marked by the red shadow, and points belonging to the convex hull are marked by the blue shadow. Notice that the blue square, representing a non-dominated point, does not belong to the convex hull because it lies in the non-convex region of the Pareto front. The gray stars in (a) and dashed lines in (b) are dominated points.

associates each solution a line [1].

Let us assume a finite state space $\mathcal{S}$, action space $\mathcal{A}$ and time horizon T in the considered MOMDP. Through an iterative optimization of all weight settings corresponding to the points on the convex hull of the current Q-vectors, the algorithm in Barrett and Narayanan [2008] narrows down the set of selected weight settings, at the expense of a higher complexity of the modified value updating procedure in standard Q-learning : the $\mathcal{O}(|A|)$ complexity ($Q(s,a)$ updating operation in Algorithm (2.3)) is multiplied by a factor $O(n^d)$, where n is the number of points on the convex hull of the Q-vectors. While the approach provides optimality guarantees (n converge toward the number of Pareto optimal policies), the number of intermediate solutions can be huge (in the worst case, n equals $|A| \cdot |S|$, which is the total number of Q-vectors maintained by the algorithm).

Noticing the equivalence between the Q-vectors (Figure 4.1(a)) and their linear scalarized values (Figure 4.1(b)), Lizotte et al. [2012] extends [Barrett and Narayanan, 2008] by representing the convex hull in piece-wise linear value functions. Under this representation, the search of points on the convex hull is carried out by merging piece-wise linear functions, which is more computationally efficient than the point searching operations in [Barrett and Narayanan, 2008]. The growth of n values is thus kept under control, which is at most $|\mathcal{A}| + 1$ for each value update.

4.4.2 Non-linear scalarization based multiple-policy algorithms

A simple way to realize the non-linear multiple-policy MORL is to repeat the single-policy MORL algorithms with varying parameters ρ in non-linear scalarization function g_ρ. Vamplew et al. [2010] demonstrate this approach by adopting multiple user defined parameters in the TLQ-learning algorithm [Gábor et al., 1998]. However, this approach requires to know explicitly the scalarization function g_ρ, which may be infeasible in some application scenarios of multiple-policy MORL (more in section 4.5.1).

4.4.3 Population-based multiple-policy algorithms

Although population-based scalarization has been extensively used in the multi-objective evolutionary algorithms (MOEAs, section 3.3), there is surprisingly little work on the multiple-policy MORL using population-based scalarization functions. To the best of our knowledge, the only work that implements non-linear scalarization function over multiple value vectors is done by Van Moffaert et al. [2013] in which the hypervolume-based Q-learning (HBQ-learning) algorithm is proposed (Algorithm 4.2). In this algorithm, an archive P of Q-vectors of all executed actions is maintained. In each iteration of the algorithm, the selected action is the one maximizing the hypervolume indicator w.r.t. the current archive P. In the case where none of the admissible actions has a positive hypervolume indicator (which is the most frequent case), the action is uniformly selected. For each $i = 1, 2, \cdots, d$, the scalar action value $Q_i(s,a)$ is updated as in the standard Q-learning approach.

[1] The concept of convex hull is also well-known in the literature on partially-observable Markov decision process (POMDP, [Smallwood and Sondik, 1973]).

Algorithm 4.2: Hypervolume-based Q-learning algorithm

Input: A multi-objective generative model $\mathcal{M}_d$, a policy π, learning rate $\alpha \in]0, 1]$
Output: Approximated action value function $\mathbf{Q}$
Initialize $Q_i(s, a)$ arbitrarily for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ and $i = 1, 2, \cdots, d$
repeat
  Select an initial $s \in \mathcal{S}$, archive $P \leftarrow \{\}$
  repeat
    $a \leftarrow \pi(s)$
    $(s', \mathbf{r}) \leftarrow \mathcal{M}_d(s, a)$
    $a'_{max} \leftarrow \arg\max_{a' \in \mathcal{A}_{s'}} HV(P \cup \{\mathbf{Q}(s', a')\}, z)$ //z is the reference point
    $\mathbf{Q}(s, a) \leftarrow (1 - \alpha)\mathbf{Q}(s, a) + \alpha[\mathbf{r} + \gamma \cdot \mathbf{Q}(s', a'_{max})]$
    $P \leftarrow P \cup \{\mathbf{Q}(s, a)\}$
    $s \leftarrow s'$
  until s is the terminal state
until stopping criterion
return $\mathbf{Q}$

The main benefit of population-based multiple-policy MORLs is that no preference information needs to be specified before the optimization. However, the drawback of these algorithms is that a point set P is required to be maintained, and the population-based scalarization function (such as the hypervolume indicator, section 3.3.3) is more computationally expensive than other non-linear scalarization functions.

4.5 MORL applications

4.5.1 Application scenarios

MORL has been employed in a wide range of applications, both in simulation and real-world settings. In this section, we classify these applications into three categories based on how the user's preferences are taken into account (Figure 4.2). All three scenarios contain a planning phase and an execution phase. In all real-world applications, no matter how many solutions are found by the MORL during the planning phase, only one policy can be executed in the execution phase.

The first scenario is called the *known preference scenario*, in which the user preference is specified in the form of scalarization function g_ρ with fixed parameters ρ at the time of planning (Figure 4.2(a)). Having an explicit presentation of scalarization function, the MOO problem is reduced to a single-objective optimization problem, and single-policy MORLs with both linear and non-linear (e.g. TLQ-learning) scalarization functions are applicable in such scenario. Notice that when the scalarization function is non-linear, the additive property of each objective does not necessarily hold for the scalarized value function, and standard RL methods can not be applied. An alternative is to use evolutionary algorithms (section 3.3.1) in this case.

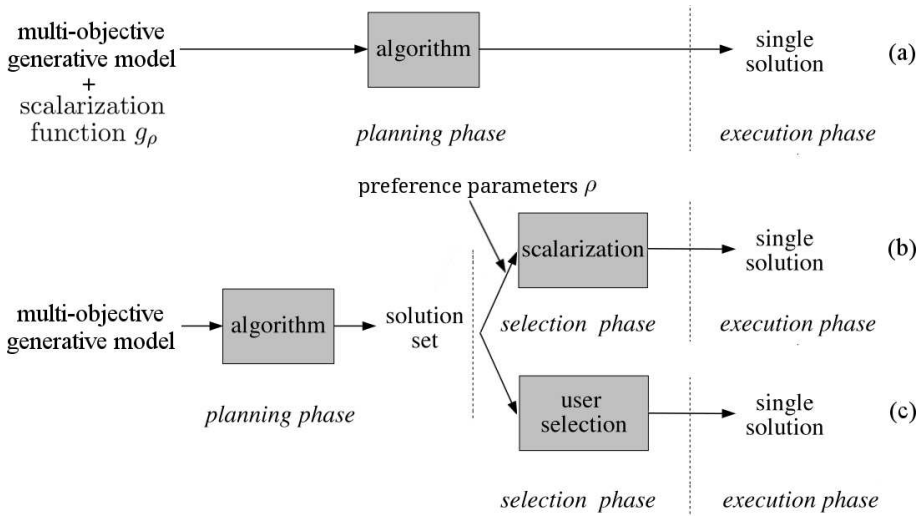


Figure 4.2: The three application scenarios for MORL. (a) Known preference scenario; (b) Varying preference scenario; (c) Decision support scenario (adapted from [Roijers et al., 2013]).

Both the second and the third scenarios start by building a Pareto optimal solution set. In the second scenario, called the *varying preference scenario* (Figure 4.2(b)), the user preferences are expressed in the scalarization function g_ρ, but the parameter ρ may change due to the current context of deployment. Consider for example a public transport system which aims to minimize both latency (i.e. the time that commuters need to reach their destinations) and pollution costs. Assume that the resulting multi-objective SDM can be scalarized by converting each objective into monetary cost: economists can impose a tax $g_\rho(\mathbf{r})$ representing the lost public welfare due to latency and pollution. Such monetary costs vary due to the change of public opinion (represented by the parameter ρ) towards latency and pollution. In such a scenario, multiply-policy MORLs can be used to compute a set of Pareto optimal policies such that, for any public preference ρ, one of those policies is optimal. When it is time to select a policy (the *selection phase*), the current preference ρ is used to determine the best policy from the solution set.

In the third scenario, called the *decision support scenario*, aggregating multiple objectives into a single one is infeasible throughout the entire decision-making process due to the difficulty of specifying the user preferences in an explicit way. For example, in the urban planning case, the decision maker usually has a subjective preference over existing construction plans, which defy precise quantification: when the construction of an intercity rail-road can be made more efficient by obstructing a beautiful view, the human designer may not be able to quantify the loss of beauty. The difficulty of specifying the exact scalarization becomes even more apparent when the designer is not a single person but a committee or legislative body which is composed of members with different preferences. In such systems, the multiple-policy MORL is used to calculate a Pareto optimal solution set. As is shown by Figure 4.2(c), in the selection phase of the decision support scenario,

the user(s) select a policy from the set of solutions according to their arbitrary preferences, rather than maximizing an explicit scalarized value function.

4.5.2 Real-world applications

This section surveys some representative applications of MORL algorithms.

Robotics is one of the earliest application fields of MORL (note that most work in this field has been in simulation). Maravall and de Lope [2002] consider the control of a robot, with the objectives of moving in a desired direction while avoiding collisions. Meisner et al. [2009] identify social robots as a promising application of MORL methods: their behaviour is inherently multi-objective because they must carry out a task without causing anxiety or discomfort for humans.

Reflecting the increasing social and political concerns on the environmental problems, many MORL algorithms have been proposed to balance the trade-off among economic, social and environmental objectives. One of the most extensively studied applications of MORL is the water reservoir control problem [Soncini-Sessa et al., 2003; Castelletti et al., 2008, 2011]. The task in this problem is to find a control policy for releasing water from a dam while balancing multiple functionalities of the reservoir, including hydroelectric and flood mitigation. Management of hydroelectric power production is also examined by Shabani [2009].

Computing and communication applications have been widely considered as well. Both Tesauro et al. [2007] and Liu et al. [2010] consider the problem of controlling a computing server, with the objectives of minimizing both response time to user requests and the power consumption. Perez et al. [2009] apply a linear scalarized single-policy MORL method to the allocation of resources to jobs in a cloud computing scenario, with the objectives of maximizing system responsiveness, utilization of resources, and fairness among users. Comsa et al. [2012] consider how to maximize system throughput and ensure user equity in the context of Longer Term Evolution (LTE) mobile communications packet scheduling protocol. Zheng et al. [2012] also use constrained MORL methods to making routing decisions in a cognitive radio network, aiming to minimize average transmission delay while maintaining an acceptably low packet loss rate.

MORL has also been applied to the control of traffic infrastructure. Yang and Wen [2010] apply it to the control of freeway on-ramps and vehicle management systems, aiming to maximize both the throughput and equity of a freeway system. Houli et al. [2010] also apply MORL to traffic light control. Their approach considers different objectives based on the current state of the road system: minimizing vehicle stops is prioritized when traffic is free-flowing; minimizing waiting time is emphasized when the system is at medium load; and minimizing queue length at intersections is targeted when the system is congested.

Lizotte et al. [2012] apply MORL in a medical application: prescribing an appropriate drug regime for a patient so as to achieve an acceptable trade-off between the effectiveness and the severity of the drug and its side effects. Their system learns multiple policies based on the data produced during randomized drug trials. The selection of the best treatment for a specific patient is then made by a doctor based on that patient's individual

circumstances [2].

Table 4.1 summarizes the above MORL approaches and their application scenarios. We find that a majority of MORL applications focus on the known preference scenario, which can be reduced to a single-objective problem once the scalarization function is known. But the varying preference scenario and decision support scenario recently starts drawing the attention of researchers [Houli et al., 2010; Castelletti et al., 2011; Lizotte et al., 2012].

Table 4.1: Summary of applications of MORL to real-world problems.

Areas	References	Application Scenario
Robot control	Maravall and de Lope [2002]	known preference
Human-robot interaction	Meisner et al. [2009]	known preference
Water reservoir control	Soncini-Sessa et al. [2003]	known preference
	Castelletti et al. [2008]	known preference
	Shabani [2009]	known preference
	Castelletti et al. [2011]	decision support
Automatic computing	Tesauro et al. [2007]	known preference
	Perez et al. [2009]	known preference
	Liu et al. [2010]	known preference
Telecommunication	Comsa et al. [2012]	known preference
	Zheng et al. [2012]	known preference
Traffic control	Yang and Wen [2010]	known preference
	Houli et al. [2010]	varying preference
Medical treatment	Lizotte et al. [2012]	decision support

[2]The work of Lizotte et al. [2012] is remotely related to the work of preference-based reinforcement learning [Fürnkranz et al., 2012; Akrour et al., 2012], which is outside the scope of this manuscript.

Part III

Contributions

Chapter 5

Multi-Objective Monte-Carlo Tree Search

In order to extend MCTS to multi-objective sequential decision making, we propose in this work the multi-objective Monte-Carlo tree search (MOMCTS) which aims at discovering multiple Pareto-optimal solutions within a single tree. In this chapter, we firstly introduce the MOMCTS framework, the core of which is a local scalarization function used in each node for action selection. Two scalar functions have been considered, respectively the hypervolume indicator (section 5.2) and the Pareto dominance reward (section 5.3). The properties of the considered action selection criteria are discussed in the end of this chapter.

5.1 Overview of MOMCTS

In each node of the MOMCTS search tree, a vectorial reward $\hat{\mathbf{r}}_{s,a} = (r_{s,a;1}, r_{s,a;2}, \cdots, r_{s,a;d})$ representing the average reward in each objective is maintained, together with the number $n_{s,a}$ of visits to the node. Each tree-walk in MOMCTS involves the same three phases as MCTS (section 2.6.1) – the selection phase, the tree building phase and the random phase. In order to adapt to the MOO setting, the modifications made in the three phases are presented in the following.

5.1.1 Selection phase

The node selection in MOMCTS depends on a scalar score, which supports a total order among nodes with multi-dimensional rewards. In this work, we propose two scores in the selection phase of MOMCTS – the hypervolume indicator and the Pareto dominance reward. Both belong to the population based scalarization class (section 4.2.3). They rely on the archive P, which maintains the vectorial rewards gathered during the MOMCTS process.

5.1.2 Tree building phase

In the tree building phase, the progressive widening (PW) and Rapid Action Value Estimation (RAVE) which are optionally used in MCTS (section 2.6.2), are regularly integrated into MOMCTS. Let us recall that PW limits the number of admissible actions of a node to an integer value $\lfloor n_{s,a}^{1/b} \rfloor$, with b equalling usually 2 or 4. The selection of the action in the tree-building phase relies on the RAVE heuristic.

5.1.3 Random phase

The random phase is carried out in the same way as in MCTS, except that in the end, a vectorial reward $\mathbf{r}$ is returned. The other modification is that the population based scalarization function (section 4.2.3) will require the archive P of the received vectorial rewards to be maintained. When the number of objectives is low ($d \leq 3$), the computation and memory resources needed to maintain the archive P are limited. With no loss of generality, dominated points are removed from the archive P.

Some additional heuristics need to be devised to maintain the scalability of the population-based scalarization approach in the many-objective setting. The extension of MOMCTS to the many-objective case is a perspective for further work.

5.1.4 MOMCTS framework

The MOMCTS framework is summarized by Algorithm 5.1. The common input for all MOMCTS algorithms include the computational budget N, the b parameter used in the progressive widening heuristic, and the generative model $\mathcal{M}_d$ of the considered multi-objective SDM problem. The value of node (s, a) noted by $g_x(s, a)$ is a population based scalarization function, with x identifying the choice of scalarization method.

In MOMCTS, the Rapid Action Value Estimation (RAVE) takes a vectorial form ($\mathbf{RAVE}(a) \in \mathbb{R}^d, a \in \mathcal{A}$). A scalarization function is therefore required to recover the total order among RAVE values as well. Likewise, a scalarization of the RAVE vectors noted by $g_{x;rave}(a), a \in \mathcal{A}$ is used with the same type of scalarization method as in $g_x(s, a)$. The detailed description of $g_x(s, a)$ and $g_{x;rave}(a)$ functions is given in section 5.2 and section 5.3.

5.1.5 Discussion

Compared to MCTS, the main modification made in MOMCTS regards the node selection step. The challenge is to extend the single-objective node selection criterion (Eq.(2.20)) to the multi-objective setting. As stated, the core of the MOO is to recover the total order among points in the multi-dimensional space. The most straightforward way of dealing with multi-objective optimization is to get back to single-objective optimization, through the use of scalarization function. MOMCTS features a population based scalarization of vectorial rewards. In contrast to the MCTS, which estimates the value of nodes according to the stationary reward distribution on a single objective, MOMCTS estimates the value of nodes with multi-dimensional rewards according to their contribution to the archive P, thus along a non-stationary setting.

Through the use of population based scalarization function, MOMCTS handles a single-objective optimization problem in each tree-walk, by repetitively searching for single solutions which bring most improvement to the quality of the solution set P. Multiple tree-walks together provide an approximated Pareto optimal solution set in MOMCTS.

An important property of MCTS is the consistency property defined as the ability of algorithm to converge towards the optimal policy when the number of tree-walk goes to infinity [Berthier et al., 2010]. The consistency property critically relies on the stationary

Algorithm 5.1: MOMCTS framework

MOMCTS
Input: number N of tree-walks
Output: search tree $\mathcal{T}$
Initialize $\mathcal{T} \leftarrow$ root node (initial state), $P \leftarrow \{\}$
for $t = 1$ **to** N **do**
  TreeWalk($\mathcal{T}, P,$ root node)
end for
return $\mathcal{T}$

TreeWalk
Input: search tree $\mathcal{T}$, archive P, node s
Output: vectorial reward $\mathbf{r}_u$
if s is not a leaf node, and $\neg(\lfloor (n_s+1)^{1/b} \rfloor > \lfloor (n_s)^{1/b} \rfloor)$ // (PW test is not triggered) **then**
  Select $a^* = \arg\max\{g_x(s,a), (s,a) \in \mathcal{T}\}$
  $\mathbf{r}_u \leftarrow$ TreeWalk($\mathcal{T}, P, (s, a^*)$)
else
  $\mathcal{A}_s = \{$ admissible actions not yet visited in $s\}$
  Select $a^* = \arg\max\{g_{x;rave}(a), a \in \mathcal{A}_s\}$
  Add (s, a^*) as child node of s
  $\mathbf{r}_u \leftarrow$ RandomWalk($P, (s, a^*)$)
end if
Update n_s, n_{s,a^*}, $\hat{\mathbf{r}}_{s,a}$ and **RAVE**(a^*)
return $\mathbf{r}_u$

RandomWalk
Input: archive P, state u
Output: vectorial reward $\mathbf{r}_u$
$\mathcal{A}_{rnd} \leftarrow \{\}$ //store the set of actions visited in the random phase
while u is not final state **do**
  Uniformly select an admissible action a for u
  $\mathcal{A}_{rnd} \leftarrow \mathcal{A}_{rnd} \cup \{a\}$
  $u \leftarrow (u, a)$
end while
$\mathbf{r}_u \leftarrow \mathcal{M}_d(u)$ //obtain the vectorial reward of the tree-walk
if $\mathbf{r}_u$ is not dominated by any point in P **then**
  Prune all points dominated by r_u in P
  $P \leftarrow P \cup \{\mathbf{r}_u\}$
end if
Update **RAVE**(a) for $a \in \mathcal{A}_{rnd}$
return $\mathbf{r}_u$

assumption, that is the fact that the reward distribution is fixed. In the case of MOMCTS, however, the population-based scalarization function relies on the Pareto archive, and thus is non-stationary. Studying the consistency of the proposed MOMCTS approach is a perspective for further work.

5.2 MOMCTS based on hypervolume indicator

5.2.1 Hypervolume indicator based value estimation

We associate to each node (s, a) in the tree the vector $\bar{\mathbf{r}}_{s,a}$ of the upper confidence bounds on its rewards:

$$\bar{\mathbf{r}}_{s,a} = \left(\hat{r}_{s,a;i} + \sqrt{c_i \ln(n_s)/n_{s,a}} \right)_{i=1}^{d} \tag{5.1}$$

with c_i the exploration vs. exploitation parameter for the i-th objective (Eq.(2.20)).

As the hypervolume indicator provides a scalar measure of solution sets in the multi-objective space, it then comes naturally to define an optimistic estimate of the value of (s, a) as the hypervolume indicator contribution associated to the upper confidence vector $\bar{\mathbf{r}}_{s,a}$ w.r.t. archive P. Let us denote

$$\Delta HV(\bar{\mathbf{r}}_{s,a}) = HV(P \cup \{\bar{\mathbf{r}}_{s,a}\}; z) - HV(P; z) \tag{5.2}$$

Although $\Delta HV(\bar{\mathbf{r}}_{s,a})$ provides a scalar value of a node (s, a) conditioned on the solutions previously evaluated, the problem is that $\Delta HV(\bar{\mathbf{r}}_{s,a})$ takes on a constant value 0 if $\bar{\mathbf{r}}_{s,a}$ is dominated by some vectorial reward in P. In order to differentiate among dominated points, the proposed approach considers the perspective projection $\bar{\mathbf{r}}^{p}_{s,a}$ of $\bar{\mathbf{r}}_{s,a}$ onto the approximated Pareto surface $\mathcal{P}$ over point set P.

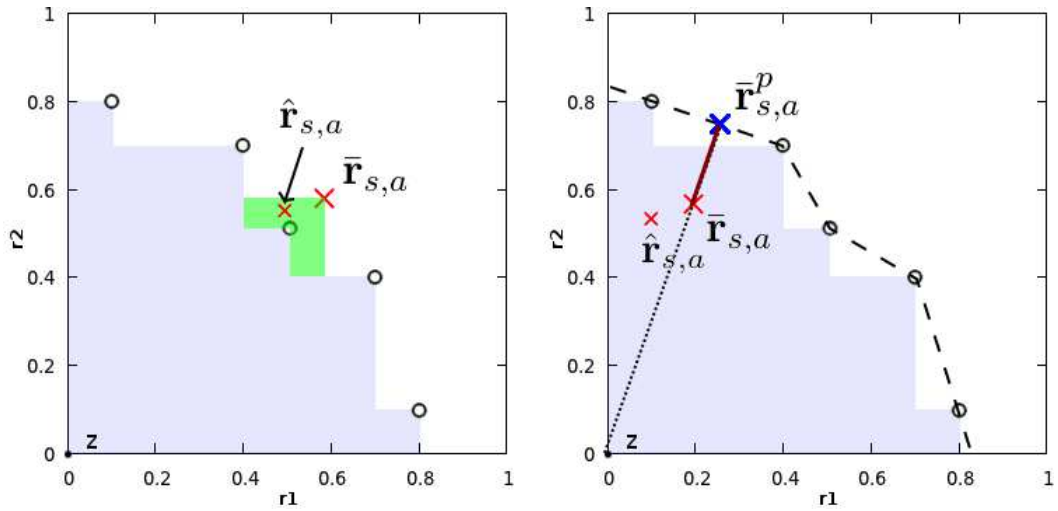
The calculation of approximated Pareto surface is one of the critical issues in MOO study [Zhou et al., 2011].

In this work, we treat $\mathcal{P}$ as the polygonal approximation of the archive P of non-dominated points. In the two-dimensional case, $\mathcal{P}$ is defined as the linear piecewise function over P (an example of this is demonstrated by the dashed lines in Figure 5.1(b)):

Definition 15. *Note P as an ordered set of two-dimensional points $\{p_1, p_2, \cdots, p_n\} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ with $x_i < x_j$ if $i < j$. Then the polygonal approximation $\mathcal{P}$ of P is the set of segments $\overline{p_i, p_{i+1}}, i = 1, 2, \cdots, n$, together with the half-lines $\overrightarrow{p_2 p_1}$ and $\overrightarrow{p_{n-1} p_n}$.*

$$\mathcal{P} = \{\overrightarrow{p_2 p_1}, \overline{p_2 p_3}, \cdots, \overline{p_{n-2} p_{n-1}}, \overrightarrow{p_{n-1} p_n})\}$$

In higher dimensional objective spaces ($d \geq 3$), the polygonal approximation of P can be obtained through several triangulation methods [Edelsbrunner and Shah, 1996; Shojaee et al., 2006], referring the reader to [Kolesnikov, 2003] for a comprehensive presentation of the surface approximation methods.



(a) Value of non-dominated vectorial reward

(b) Value of dominated vectorial reward

Figure 5.1: Left: For a vectorial reward $\bar{\mathbf{r}}_{s,a}$ that is not dominated w.r.t. the archive P, $V_{hv}(s,a)$ is its hypervolume indicator contribution to the solution set. Right: For a vectorial reward that is dominated by some point in the archive P, its value is measured by the opposite of its Euclidean distance to the approximated surface (dashed lines) of the Pareto front.

Let $\bar{\mathbf{r}}^{p}_{s,a}$ denote the unique intersection of half-line $\overrightarrow{z\ \bar{\mathbf{r}}_{s,a}}$ with $\mathcal{P}$ (being reminded that z is dominated by all points in P and by $\bar{\mathbf{r}}_{s,a}$). The value of node (s,a) is then defined as the opposite of the Euclidean distance between $\bar{\mathbf{r}}_{s,a}$ and $\bar{\mathbf{r}}^{p}_{s,a}$. Finally, the scalarization of $\bar{\mathbf{r}}_{s,a}$ is defined as:

$$g_{hv}(s,a) = \begin{cases} \Delta HV(\bar{\mathbf{r}}_{s,a}) & \text{if } \bar{\mathbf{r}}_{s,a} \text{ is non-dominated in } P \\ -\parallel \bar{\mathbf{r}}^{p}_{s,a} - \bar{\mathbf{r}}_{s,a} \parallel_2^d & \text{otherwise} \end{cases} \tag{5.3}$$

The Euclidean distance term sets a penalty for dominated points, increasing with their distance to the approximated Pareto surface $\mathcal{P}$. This term is elevated to the d-th power by homogeneity with the hypervolume indicator. Note that Eq.(5.3) sets a total order on all vectorial rewards in $\mathbb{R}^d$, where non-dominated points are ranked higher than dominated ones.

5.2.2 MOMCTS-hv algorithm

We refer to the MOMCTS algorithm based on the hypervolume indicator as MOMCTS-hv. It is realized by using

$$a^* = \arg\max g_{hv}(s,a) \tag{5.4}$$

as node selection rule in the MOMCTS framework (Algorithm 5.1).

RAVE vectors are used to select new nodes in the tree-building phase of MOMCTS-hv. Letting $\mathbf{RAVE}(a)$ denote the average vectorial reward associated to a. As $\mathbf{RAVE}(a)$ is a weighted sum over vectorial rewards gathered in previous tree-walks, they are likely to be dominated by points in the current archive P. Therefore, we scalarize RAVE vectors based on their Euclidean distance to $\mathcal{P}$. Let $\mathbf{RAVE}^p(a)$ denote the perspective projection of $\mathbf{RAVE}(a)$ on $\mathcal{P}$, then the action selected is the one maximizing

$$g_{hv;rave}(a) = - \parallel \mathbf{RAVE}^p(a) - \mathbf{RAVE}(a) \parallel_2 \tag{5.5}$$

Beside the standard MOMCTS parameters – the total number of tree-walks N, the b parameter in the progressive widening heuristic, and the generative model $\mathcal{M}_d$, MOMCTS-hv algorithm involves the following additional parameters: (i) the exploration vs. exploitation trade-off parameter c_i for every i-th objective, and (ii) the reference point z.

5.2.3 Discussion on MOMCTS-hv

Let B denote the average branching factor in the MOMCTS tree, and let N denote the number of tree-walks. As each tree-walk adds a new node, the number of nodes in the tree is $N+1$ by construction. The average length of a tree-path thus is in $\mathcal{O}(\log N)$. Depending on the number d of objectives, the hypervolume indicator is computed with complexity $\mathcal{O}(|P|^{d/2})$ for $d > 3$ (respectively $\mathcal{O}(|P|)$ for $d = 2$ and $\mathcal{O}(|P| \log |P|)$ for $d = 3$) [Beume et al., 2009]. The complexity of each tree-walk thus is $\mathcal{O}(B|P|^{d/2} log N)$, where $|P|$ is at most the number N of tree-walks.

By construction, the hypervolume indicator based selection criterion (Eq. (5.3)) drives MOMCTS towards the Pareto front and favours the diversity of the Pareto archive [Beume et al., 2007]. On the negative side however, this criterion suffers from three drawbacks:

1) The hypervolume indicator is not invariant under monotonous transformation of objective functions, which prevents the approach from enjoying the same robustness as comparison-based optimization approaches [Hansen, 2006].

2) The MOMCTS critically depends on its hyper-parameters. The exploration vs. exploitation (EvE) trade-off parameters $c_i, i = 1, 2, \ldots, d$ of each objective (Eq.(5.1)) have a significant impact on the performance of MOMCTS-hv (likewise, the MCTS applicative results depend on the tuning of the EvE trade-off parameters [Chaslot et al., 2008b]). Additionally, the choice of the reference point z also influences the hypervolume indicator values [Auger et al., 2009].

3) The computational cost of $g_{hv}(s, a)$ is exponential with the number d of objectives.

5.3 MOMCTS based on Pareto dominance reward

Aimed at overcoming the limitations in the node selection criterion based on hypervolume indicator, this section presents a new selection criterion based on the Pareto dominance test.

5.3.1 Cumulative discounted dominance reward based value estimation

The hypervolume indicator based node value estimation requires the information of the average rewards $\hat{r}_{s,a;i}$ and EvE constant c_i in each objective $i = 1, 2, \cdots, d$. Instead of gathering and updating the multi-dimensional vectorial rewards in each node, a simpler option is to associate to each tree-walk a reward 1 if the tree-walk gets a vectorial reward $\mathbf{r}_u$ which is not dominated by any point in the archive P, and reward 0 otherwise. Formally this boolean dominance reward, called $r_{u;dom}$, is defined as:

$$r_{u;dom} = \begin{cases} 1 & \text{if } \nexists \mathbf{r} \in P, \mathbf{r} \succ \mathbf{r}_u \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

The advantage of this option is its simplicity. Given a vectorial reward, the dominance reward is calculated within time $\mathcal{O}(d|P|)$. The drawback of this option is that, due to the rarity of non-dominated rewards, most tree-walks get a 0 dominance reward. Considering the the rarity of dominance rewards, the update of $r_{u;dom}$ proceeds along a cumulative discounted (CD) process as follows. Let $t_{s,a}$ denote the index of the last tree-walk which visited node (s, a), let $\Delta t = t - t_{s,a}$ where t is the index of the current tree-walk, let $\delta \in [0, 1]$ be a discount factor, the cumulative discounted dominance (CDD) reward update is defined as:

$$\begin{aligned} &\hat{r}_{s,a;dom} \leftarrow \hat{r}_{s,a;dom} \cdot \delta^{\Delta t} + r_{u;dom},\ \delta \in [0, 1] \\ &t_{s,a} \leftarrow t; \quad n_{s,a} \leftarrow n_{s,a} + 1; \quad n_s \leftarrow n_s + 1 \end{aligned} \tag{5.7}$$

The update procedure of dominance reward differs from the standard scheme (Eq.(2.21)) in two respects. Firstly, cumulative instead of average rewards are considered. The rationale for this modification is that a tiny percentage of the tree-walks finds a non-dominated vectorial reward. In such cases, average rewards come to be negligible in front of the exploration term, making the MCTS degenerate to pure random search. The use of cumulative rewards instead tends to prevent this degradation.

Secondly, a discount mechanism is used to moderate the cumulative effects using the discount factor δ ($0 \leq \delta < 1$) and taking into account the number Δt of tree-walks since this node was last visited. This discount mechanism is meant to cope with the dynamics of multi-objective search through forgetting old rewards, thus enabling the decision rule to reflect up-to-date information.

Indeed, the CD process is reminiscent of the discounted cumulative reward defining the value function in Reinforcement Learning [Sutton and Barto, 1998], with the difference that the time-step t here corresponds to the tree-walk index, and that the discount mechanism is meant to limit the impact of past (as opposed to, future) information.

In a stationary context, suppose that the node (s, a) corresponds to an expectation of dominance reward $\mathbb{E}_r = \mathbb{E}(r_{s,a;dom}) \in [0, 1]$, and the interval of time between two visits to the node is fixed as Δt, then $\hat{r}_{s,a;dom}$ would converge towards $\frac{1}{1-\delta^{\Delta t}}\mathbb{E}_r$. If the node gets rarely visited ($\Delta t \gg 1$), then $\delta^{\Delta t}$ goes to 0 and $\hat{r}_{s,a;dom}$ goes to the average reward $\mathbb{E}_r$. Quite the contrary, if the node happens to be frequently visited ($\Delta t = 1$), the cumulative reward equals the reward expectation $\mathbb{E}_r$ multiplied by a large factor ($\frac{1}{1-\delta}\mathbb{E}_r$), entailing

the over-exploitation of the node. However, the over-exploitation is bound to decrease as soon as the Pareto archive moves towards the true Pareto front, which reduces the reward expectation $\mathbb{E}_r$. In section 5.4.3, the CDD reward properties are illustrated through an example problem.

Finally, the node value estimation based on the CDD rewards is defined as:

$$g_{dom}(s,a) = \hat{r}_{s,a;dom} + \sqrt{c_e \ln(n_s)/n_{s,a}} \tag{5.8}$$

5.3.2 MOMCTS-dom algorithm

We call the MOMCTS based on CDD rewards as MOMCTS-dom. It proceeds as standard MCTS except that the selection rule is defined by

$$a^* = \arg\max g_{dom}(s,a) \tag{5.9}$$

Likewise, letting $g_{dom;rave}(a)$ denote the CDD reward gathered in each action a (updated in the same way as $\hat{r}_{s,a;dom}$, Eq.(5.7)), then the selected action in the tree building phase is the one maximizing $g_{dom;rave}(a)$.

Keeping the same notations B, N and $|P|$ as above, as the dominance test in the end of each tree-walk is linear ($\mathcal{O}(d|P|)$), the complexity of each tree-walk in MOMCTS-dom is $\mathcal{O}(B \log N + d|P|)$, thus linear w.r.t. the number d of objectives.

Besides the common MOMCTS parameters N, b and $\mathcal{M}_d$, MOMCTS-dom involves two additional hyper-parameters: i) the exploration vs. exploitation trade-off parameter c_e; and ii) the discount factor δ.

Compared to MOMCTS-hv, MOMCTS-dom enjoys a smaller computational complexity. The price to pay for the improved scalability of MOMCTS-dom is that the dominance reward might less favour the diversity of the Pareto archive than the hypervolume indicator: any non-dominated point has the same dominance reward whereas the hypervolume indicator contribution of non-dominated points in sparsely populated regions of the Pareto archive is higher.

5.4 Proof of concept

Within the MOMCTS framework defined in section 5.1.4, let us study the behaviour of hypervolume indicator and dominance reward based selection rules on an artificial problem.

5.4.1 Example problem

In the following, we examine the properties of hypervolume indicator and CDD rewards through a bi-objective MAB problem.

Let us define a MAB problem with 3 arms $\mathcal{A} = \{1, 2, 3\}$, each arm $a \in \mathcal{A}$ brings a vectorial reward (r_a, r'_a). Each reward r_a (respectively r'_a) is an uniformly distributed

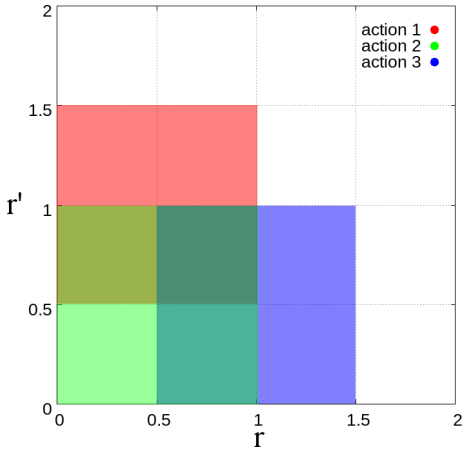


Figure 5.2: Reward distributions of the bi-objective MAB problem. The reward regions corresponding to action $1, 2$ and 3 are respectively marked by green, blue and purple shadows.

random variable in $[l_a, u_a]$ (resp. $[l'_a, u'_a]$). Figure 5.6 shows the reward distribution of actions in $\mathcal{A}$ with $r_1 \in [0, 1], r'_1 \in [0.5, 1.5]; r_2 \in [0, 1], r'_2 \in [0, 1]; r_3 \in [0.5, 1.5], r'_3 \in [0, 1]$.

Noting the expected value of actions $a \in \mathcal{A}$ by $\mathbb{E}_a = (\mathbb{E}(r_a), \mathbb{E}(r'_a))$, we have $\mathbb{E}_1 = (0.5, 1), \mathbb{E}_2 = (0.5, 0.5)$ and $\mathbb{E}_3 = (1, 0.5)$ with the same variance in each action. It is easy to see that a_1 and a_3 are non-dominated, and both dominate a_2.

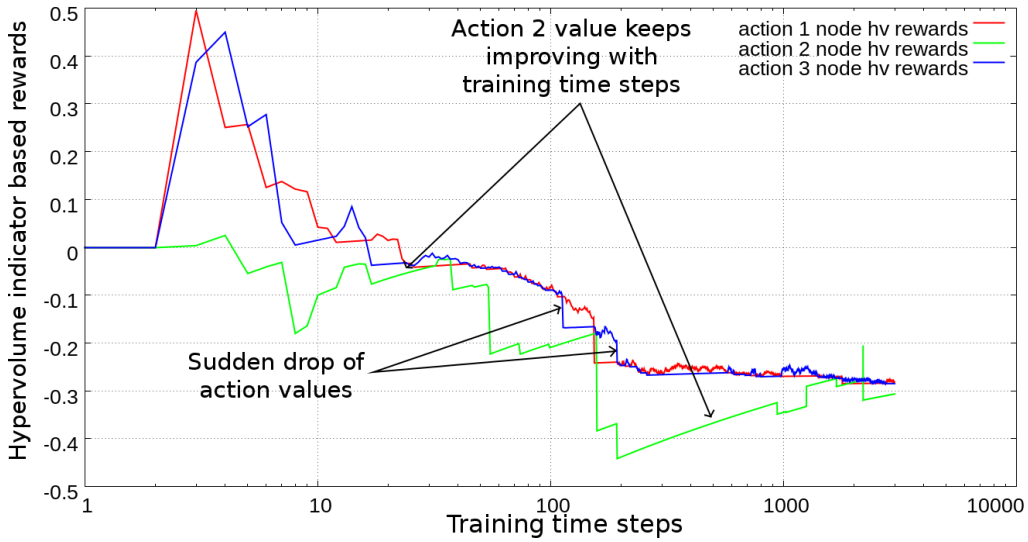
5.4.2 Hypervolume indicator based criterion analysis

Figure 5.3(a) and (b) respectively show the hypervolume indicator scores ($g_{hv}(s, a)$) of actions along time for various EvE trade-off parameters. For the low values of the exploration terms ($c_r = c_{r'} = 0.1$, Figure 5.3(a)), two phenomena are observed. Firstly, it is found that the hypervolume indicator score of action 2 tends to improve, while the hypervolume indicator scores of action 1 and 3 tend to fall with training time steps. Such phenomenon can be explained by examining Eq.(5.1):

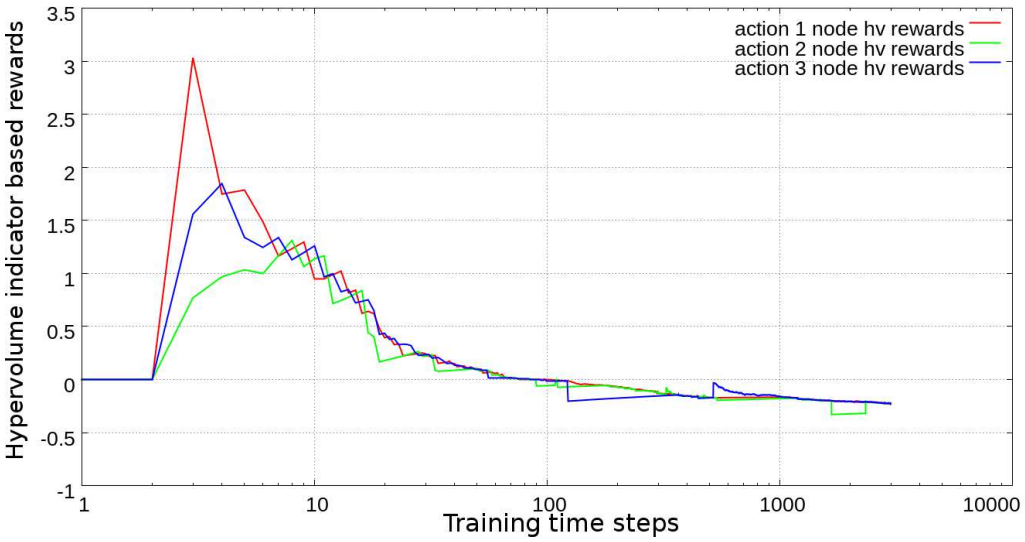
$$\bar{\mathbf{r}}_a = \left(\hat{r}_{a;i} + \sqrt{c_i \ln(n)/n_a}\right)_{i=1}^{d}$$

where n_a is the number of play times in each action, and $n = \sum_{a \in \mathcal{A}} n_a$. As action 2 is rarely played, n_2 stays unchanged during most periods of the searching process. Then the increase of the total number n of action selections result in the augmentation of the exploration terms ($\sqrt{c_i \ln(n)/n_2}, i = 1, 2$) of action 2, on the top of which the hypervolume indicator score is computed. This explains the tendency that action 2 values keeps improving throughout the searching process. Quite the contrary, because action 1 and 3 are frequently played, their n_a values increase more rapidly than the $\ln(n)$ term. Then the exploration terms of action 1 and 3 gradually drop to 0 when n goes to infinity.

Secondly, some abrupt changes appear in the evolution curves of the hypervolume indicator score. A tentative interpretation for this fact is as follows. A first mechanism



(a) $c_r = c_{r'} = 0.1$



(b) $c_r = c_{r'} = 1$

Figure 5.3: The evolution of hypervolume indicator rewards in action $1, 2, 3$ of representative runs under different EvE trade-off parameter settings.

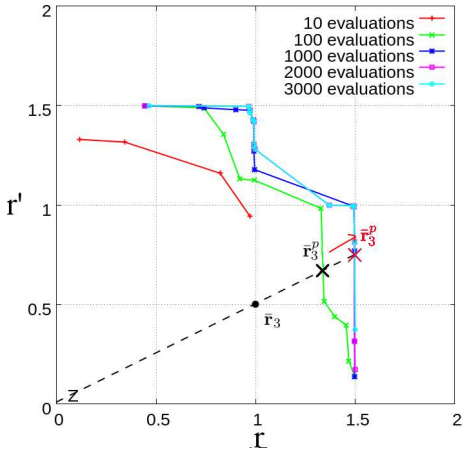


Figure 5.4: The approximated Pareto surface found under the hypervolume indicator based action selection criterion with $c_r = c_{r'} = 0.1$. The evolution of the empirical Pareto front has a non-smooth impact on the hypervolume indicator base value estimation (e.g. $\bar{\mathbf{r}}_3^p$ jumps forward when the Pareto front moves).

is that the action vectorial reward $\hat{\mathbf{r}}_a$ becomes more stable as action a is played for more times. On the other hand, the hypervolume indicator reflects the changes in the Pareto archive P as new non-dominated solutions are discovered. The hypervolume indicator score associated to each action is computed in each time step. As is shown by Figure 5.4, the distance between $\hat{\mathbf{r}}_a$ and the Pareto front may change abruptly when new non-dominated solutions are found. The score of action 2 abruptly decreases in the beginning as non-dominated solutions are discovered when triggering action 1 and 3. After a while, the score of all three actions become negative, as the Pareto archive P better approximates the true Pareto front (Figure 5.4). For action 1 and 3, the hypervolume indicator score converges to the distance between their average vectorial reward $\bar{\mathbf{r}}_a$ and the true Pareto front.

For the high values of the exploration terms ($c_r = c_{r'} = 1$, Figure 5.3(b)), the gap between the three actions are shorten and the dominated action 2 are played more frequently than in the low exploration case. Eventually, the number of times (frequency) that each action are selected during 3000 time steps under different experimental settings are summarized by Table 5.1.

5.4.3 CDD reward analysis

By implementing the CDD reward based action selection criterion (Eq.(5.8)) with $\delta = 0.95$ and $c_e = 1$ for 3000 time steps, we obtain the curve of of dominance reward evolution in action $1, 2$ and 3 in Figure 5.6(a). It is observed that, successive discovery of non-dominated rewards in action 3 initially renders this action over explored, and no other actions are played between time step 15 and 50. However, as the discovery of non-dominated rewards in action 3 becomes more rare, the dominance score associated action 3 exponentially

Table 5.1: Action selection frequencies among 3000 time steps in the bi-objective MAB problem (averaged over 11 runs).

Selection criterion	Parameter setting	Action 1	Action 2	Action 3
hypervolume indicator reward	$c_r = c_{r'} = 0.1$	1505.9±1225.6	38.1±56.2	1456.0±1206.2
	$c_r = c_{r'} = 1$	1386.5±676.2	159.2±67.0	1458.8±685.1
CDD reward	$\delta = 0.9, c_e = 1$	1012.8±29.0	836.3±266.7	1060.0±48.8
	$\delta = 0.95, c_e = 1$	1094.3±77.3	819.9±67.1	1085.9±81.5
	$\delta = 0.99, c_e = 1$	1168.8±248.7	498.8±117.1	1331.5±291.2
	$\delta = 0.999, c_e = 1$	1479±1339.2	3.5±2.0	1350.2±1346.8

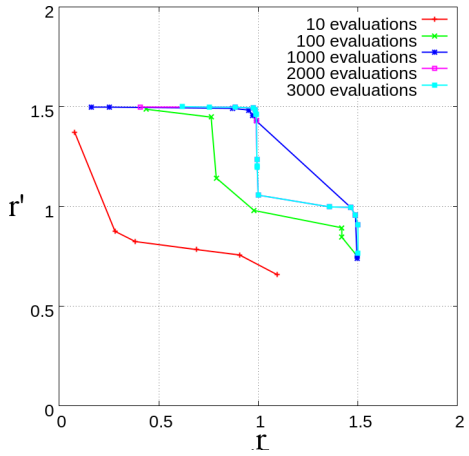
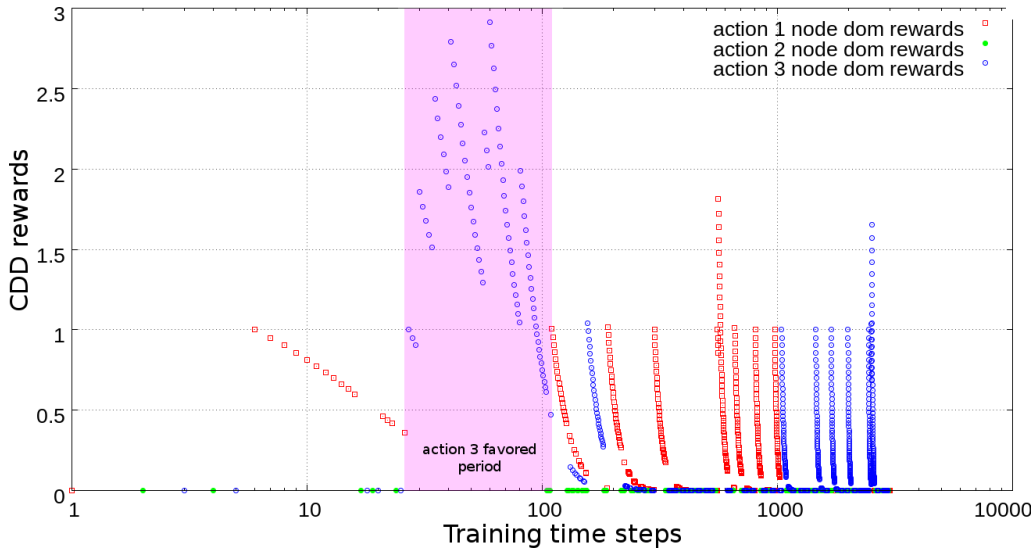


Figure 5.5: The Pareto optimal solution set found under the CDD based action selection criterion gradually moves towards the true Pareto front.

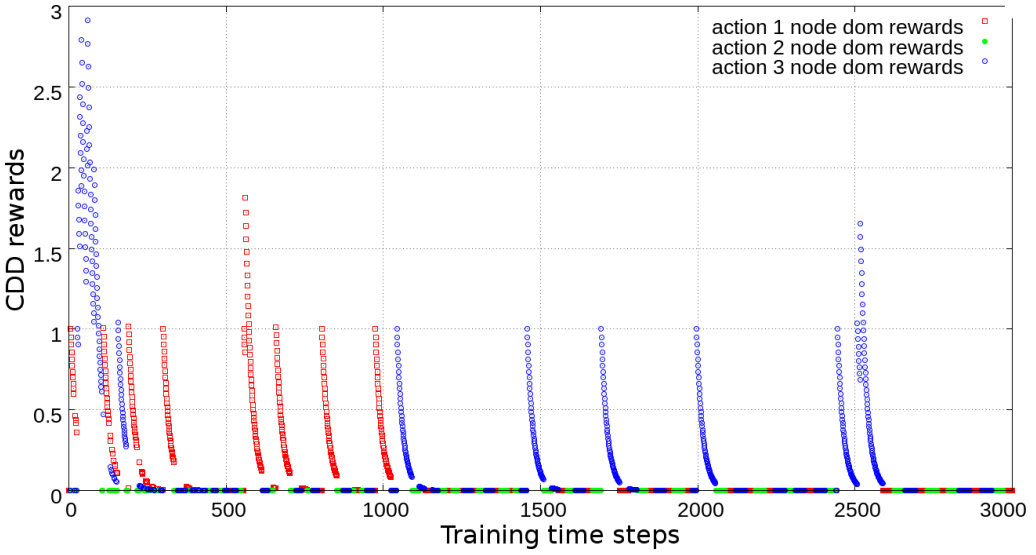
drops down due to the discounting phenomenon. On the other hand, the exploration term triggers other arms and the algorithm switches to action 1 and 2 after time step 50, and very soon discards action 2.

As could have been expected, the discovery of non-dominated rewards becomes more rare, and the interval between two successive discoveries of non-dominated rewards becomes longer (Figure 5.6(b)).

Table 5.1 shows the influence of the discount factor δ over action frequencies. It is found that the performance of the algorithm is influenced by the δ parameter from two aspects. Firstly, for greater δ values, as the cumulative effect of dominance reward lasts longer, the non-dominated actions 1 and 3 are played more frequently (reflected by the average frequency values) than in the smaller δ case. Secondly, it is observed that greater δ values result in greater variances of action frequencies due to the unbalanced exploration. For example, when $\delta = 0.999$, it is often the case that action 1 is played for more than 2800 times while action 3 is played for only 100 times, or inversely. Therefore, a careful tuning of the δ value is required for the discovery of all non-dominated solutions.



(a) Log scale in x-axis



(b) Normal scale in x-axis

Figure 5.6: The evolution of CDD rewards in action $1, 2, 3$.

5.4.4 Discussion

By construction, and as confirmed by the proof of preliminary experiment, both hypervolume indicator and CDD reward based selection rules support the discovery of the true Pareto front.

The main difference between the hypervolume indicator and CDD reward based action selection criteria lies in the fact that, all vectorial reward have the same impact on the average reward $\bar{\mathbf{r}}_a$ and on the hypervolume indicator scores, no matter when they are discovered. Quite the contrary, the CDD reward depends on the dynamics of the search.

Chapter 6

Experimental Analysis

This chapter presents the experimental validation of MOMCTS. Two artificial problems and two real-world applications are considered in our experiments to assess the performance of MOMCTS in multi-objective SDM problems with convex and non-convex Pareto front, deterministic and probabilistic transition functions, many objectives ($d \geq 3$) and real-time decision making settings. The real-time problem was considered in the framework of 2013 Multi-Objective Physical Travelling Salesman Problem (PTSP) competition where MOMCTS-based controller got the 2nd rank.

6.1 Goals of experiments

The experiments in this manuscript are carried out with two goals in mind. The first goal is to assess the performance of the MOMCTS approaches comparatively to the state of the art in MORL [Vamplew et al., 2010]. Two artificial benchmark problems (Deep Sea Treasure and Resource Gathering) with probabilistic transition functions are considered to this aim. The Deep Sea Treasure problem has two objectives which define a non-convex Pareto front (section 6.2). The Resource Gathering problem has three objectives and a convex Pareto front (section 6.3).

The second goal is to assess the performance and scalability of MOMCTS approaches in real-world setting. The MOMCTS algorithm calculates multiple Pareto optimal policies in multi-objective SDM problems. Depending on whether the user preference function is explicitly known in the execution phase of policies, MOMCTS can be implemented in two different application scenarios – the decision support scenario and the varying preference scenario (section 4.5.1). In order to assess the MOMCTS performance in these scenarios, we firstly test MOMCTS in the grid scheduling problem which is a representative application in the decision support scenario. Then the physical travelling salesman problem (PTSP) is used to assess the performance of MOMCTS in varying preference scenario with real-time constraints. We consider the Physical Travelling Salesman Problem as we participated in the international competition of multi-objective Physical Travelling Salesman Problem during the 2013 Computational Intelligence in Games (CIG) conference. In contrast to artificial problems, baseline algorithms considered in the real-world experiments of this work do not include MORL algorithms. This is due to the fact that the state space in real-world problems are unknown in advance.

The considered experiments are summarized in Table 6.1. All reported results in the experiments are averaged over 11 runs unless stated otherwise. The quality of solution sets in all experiments are measured by the hypervolume indicator, generational distance (GD) and inverted generational distance (IGD) (section 3.3.3) w.r.t. the reference Pareto

Table 6.1: Multi-objective SDM problems

Problem	Convex or Non-convex Pareto front	Deterministic or Stochastic transition function	Number of objectives	Real-time decision
Deep Sea Treasure	Non-convex	Deterministic and Stochastic	2	No
Resource Gathering	Convex	Stochastic	3	No
Grid Scheduling	Unknown	Deterministic	2	No
Physical Travelling Salesman	Unknown	Deterministic	7	Yes

front [1]. The algorithms are also assessed w.r.t. their computational cost (measured on a PC with Intel dual-core CPU 2.66GHz).

6.2 Deep Sea Treasure problem

The choice of Deep Sea Treasure (DST) problem is motivated as it involves a non-convex Pareto front. As already discussed, the challenge with non-convex Pareto front is that the non-convex regions can not be discovered using linear-scalarization methods, and the discovery of these regions remains a main challenge for MOO approaches [2]. It must also be emphasized that the chance for the Pareto front to be convex decreases with the number of objectives, everything else being equal.

Additionally, we investigate the impact of non-deterministic transition functions on the DST context.

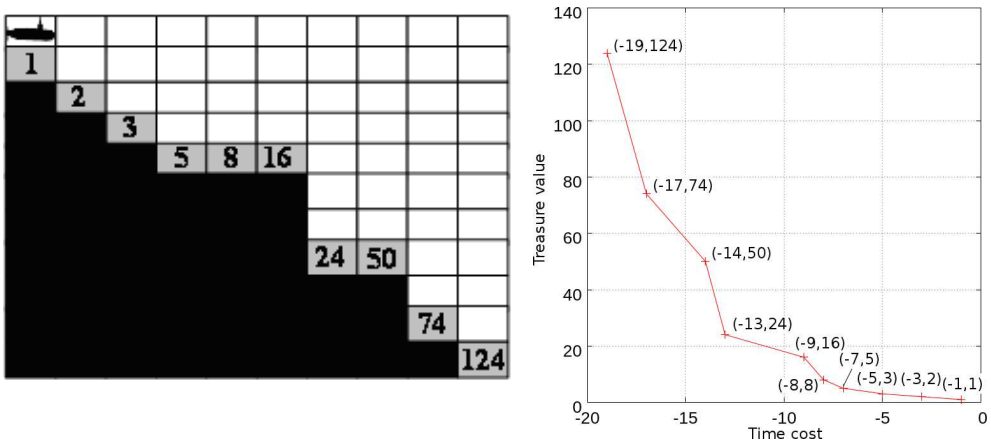
6.2.1 Problem statement

The Deep Sea Treasure (DST) problem was first introduced by Vamplew et al. [2010]. The state space of DST consists of a 10×11 grid (Figure 6.1(a)). The action space of DST includes four actions (up, down, left and right), each sending the agent to one adjacent square in the indicated direction. When the agent would go beyond the border line of the grid or touch the sea floor, it stays in the same place. Each policy, with the top left square as initial state, gets a two dimensional reward: the time spent until reaching a terminal state or reaching the time horizon $T = 100$, and the treasure attached to the reached terminal state if any (depicted in Figure 6.1(a)), or 0 otherwise. The 10 non-dominated vectorial rewards in the form of (-time,treasure) are depicted in the two-dimensional plane in Figure 6.1(b), forming a non-convex Pareto front.

In our experiments, the transition model of the DST is modified and converted into a stochastic model as follows. When action is executed, the agent would go to the indicated

[1] The reference Pareto front is the true Pareto front when it is known; otherwise, it is set to the union of all non-dominated solutions found over all runs and all algorithms.

[2] Examples of such MOO with non-convex Pareto front are ZDT2 and DTLZ2 test benchmarks [Deb et al., 2002].



(a) The state space

(b) The Pareto front

Figure 6.1: The Deep Sea Treasure problem. Left: the DST state space with black cells as sea-floor, gray cells as terminal states, the treasure value is indicated in each cell. The initial position is the upper left cell. Right: the Pareto front in the time×treasure plane.

direction with probability $1 - \eta$, and in the other directions with equal probability $\eta/3$, where $0 \leq \eta < 1$ indicates the noise level in the environment.

6.2.2 Experimental setting

In the DST problem, the performance of MOMCTS-hv and MOMCTS-dom are compared with that of the baseline algorithm – MOQ-learning (section 4.4.1), with the same parameters as in [Vamplew et al., 2010]:

- ϵ-greedy exploration is used with $\epsilon = 0.1$.
- Learning rate α is set to 0.1.
- The state-action value table is optimistically initialized ($time = 0, treasure = 124$).
- Due to the episodic nature of DST, no discounting is used in MOQ-learning ($\gamma = 1$).
- In the bi-objective DST problem, the number m of weight settings ranges in $\{3, 7, 21\}$, with $\lambda_i = \frac{i-1}{l-1}, i = 1, 2, \ldots, m$.

A few preliminary experiments are used to adjust the parameters in MOMCTS. The progressive widening parameters b is set to 2 in both MOMCTS-hv and MOMCTS-dom. In MOMCTS-hv, the exploration vs. exploitation (EvE) trade-off parameters in the time cost and treasure value objectives are respectively set to $c_{time} = 20,000$ and $c_{treasure} = 150$. In MOMCTS-dom, the EvE trade-off parameter c_e is set to 1, and the discount factor δ is set to 0.999.

As the DST problem is concerned with minimizing the search time (maximizing its opposite) and maximizing the treasure value, the reference point used in the hypervolume

indicator calculation is set to (-100,0). The hypervolume indicator of the Pareto optimal solution set is 10455.

Experiments are carried out in a DST simulator with the η noise level ranging in $0, 0.001, 0.01, 0.05$ and 0.1. Each run of MOQ-learning, MOMCTS-hv and MOMCTS-dom is limited to 300,000 time steps (ca 37,000 tree-walks in MOMCTS-hv and 45,000 tree-walks in MOMCTS-dom [3]). The entire training process is equally divided into $n_{tr} = 150$ phases. At the end of the i-th training phase, the MOQ-learning and MOMCTS solution sets are tested in the DST simulator, and form the Pareto set P_i. The performance of algorithms is reported as the hypervolume indicator of P_i.

6.2.3 Results

Table 6.2: The DST problem: hypervolume indicator results of MOMCTS-hv, MOMCTS-dom and MOQ-learning with m ranging in 3,7 and 21 with different noise levels η, averaged over 11 independent runs after 300,000 time steps. The optimal hypervolume indicator is 10455. For each η, best results are indicated in bold font (significance value $p < 0.05$ according to the Student's t-test).

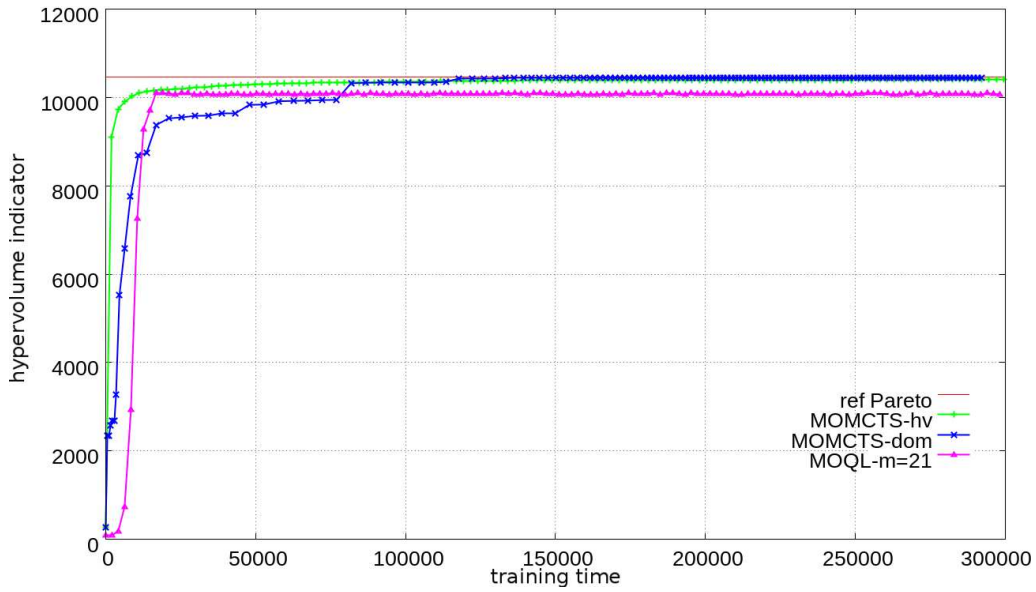
	$\eta = 0$	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.05$	$\eta = 0.1$
MOMCTS-hv	**10416±37**	**10434±31**	10436±32	10205±211	9883±1091
MOMCTS-dom	**10450±4**	**10446±19**	10389±65	9858±1153	9982±360
MOQ-learning-m=3	7099±3926	8116±3194	6422± 4353	7333±4411	6953±3775
MOQ-learning-m=7	10078±34	10049±94	9495±1701	8345±2887	8924±2663
MOQ-learning-m=21	10078±17	10085±129	7806±1933	8744±2070	6744±2355

The performance of MOMCTS approaches and MOQ-learning measured by the hypervolume indicator are reported in Table 6.2.

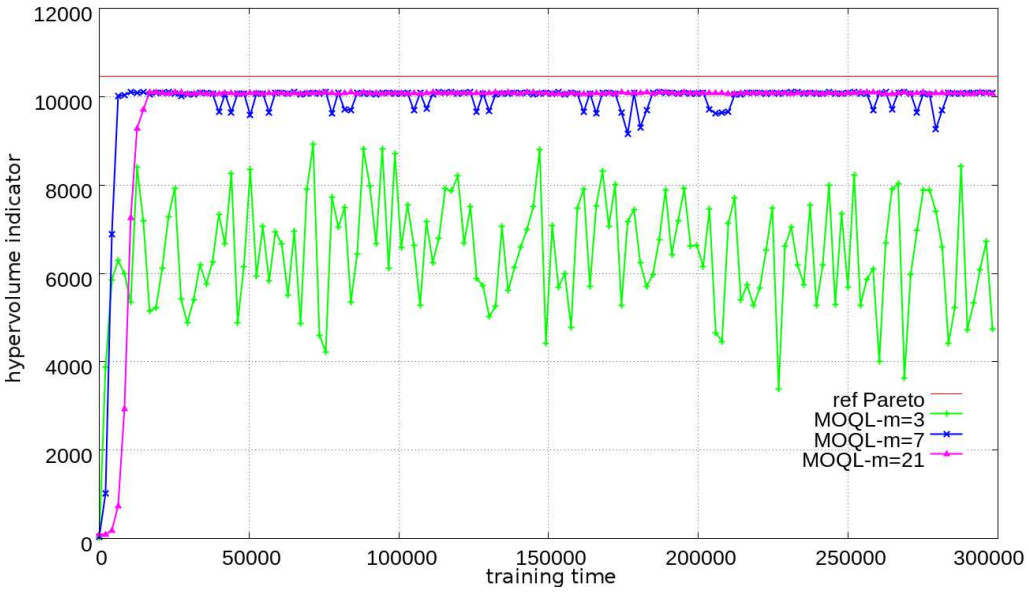
Deterministic setting Figure 6.2 displays the hypervolume indicator performance of MOMCTS-hv, MOMCTS-dom and that of MOQ-learning for $m = 3, 7, 21$ in the DST problem. It is observed that for $m = 7$ or 21, MOQ-learning reaches a performance plateau (10062) within 20,000 time steps. The fact that MOQ-learning does not reach the optimal hypervolume indicator 10455 is explained as the DST Pareto front is not convex (Figure 6.1(b)). This confirms experimentally that linear-scalarization approaches do not discover non-dominated solutions lying in the non-convex regions of the Pareto front, establishing the inconsistency of MOQ-learning.

MOMCTS-hv features a very fast convergence towards the true Pareto front, dominating all other approaches. However, it finds the full Pareto front in 5 out of 11 runs. MOMCTS-dom is slow to catch up MOMCTS-hv and MOQ-learning (after 80,000 time steps), but it ultimately outperforms MOMCTS-hv (after approximately 120,000 time

[3] Due to different node selection criteria, the average tree depths in MOMCTS-hv and MOMCTS-dom are different.



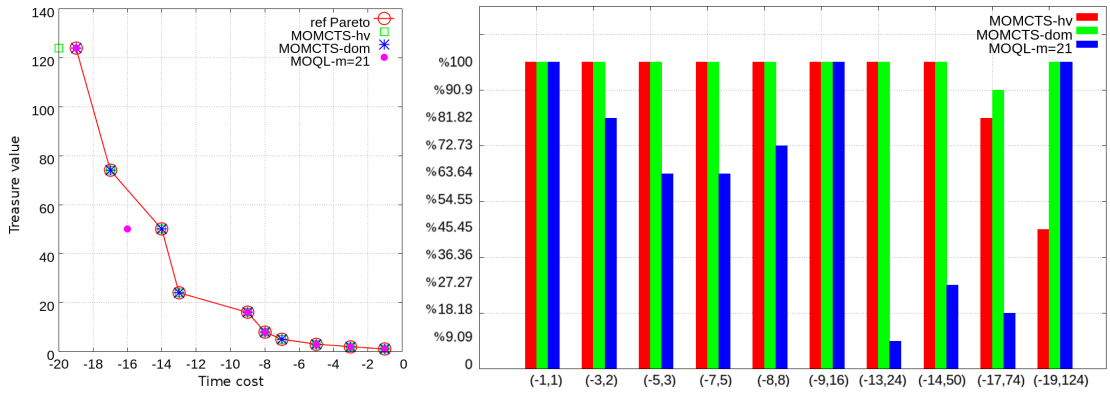
(a)



(b)

Figure 6.2: The hypervolume indicator performance of MOMCTS-hv, MOMCTS-dom and MOQ-learning versus training time in the deterministic DST problem. For the sake of a fair comparison with MOQ-learning, the training time refers to the number of action selections in MOMCTS approaches (each tree-walk in MOMCTS carries out on average 7 to 8 action selections in the DST problem). Top: The hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning-m=21. Bottom: The hypervolume indicator of MOQ-learning with $m = 3, 7, 21$.

steps), and reaches the entire Pareto front in 10 out of 11 runs. The instance of MOQ-learning is analysed in Figure 6.2(b). The general trend is that the initial progress is faster for low value of m, the price to pay is the instability of the hypervolume indicator performance, which only disappears for $m = 21$.



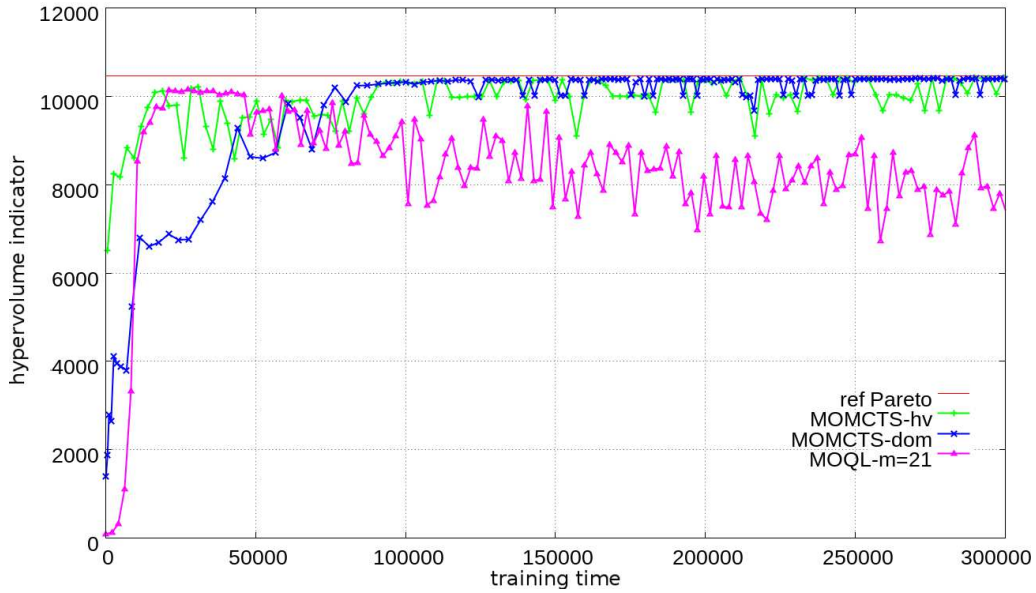
(a) (b)

Figure 6.3: Left: The vectorial rewards found by representative MOMCTS-hv, MOMCTS-dom and MOQ-learning-$m = 21$ runs. Right: The percentage of times out of 11 runs that each non-dominated vectorial reward was discovered by MOMCTS-hv, MOMCTS-dom and MOQ-learning-$m = 21$, during at least one test episode.

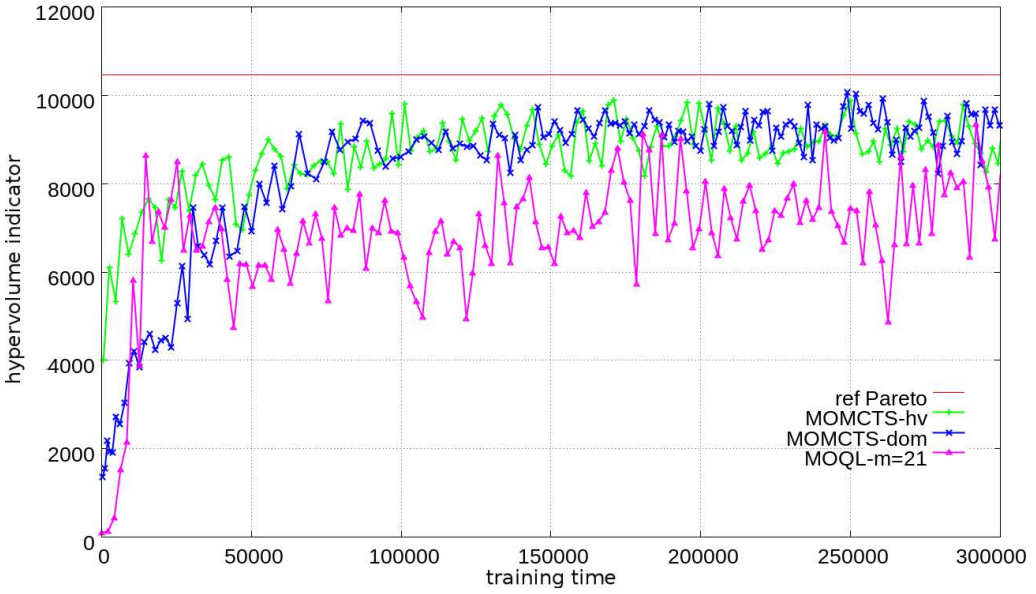
The percentage of times out of 11 runs that each non-dominated vectorial reward is discovered for at least one test episode during the training process of MOMCTS-hv, MOMCTS-dom and MOQ-learning for $m = 21$ is displayed in Figure 6.3(b). Figure 6.3 shows that MOQ-learning discovers all strategies (lying in the non-convex regions of the Pareto front) during intermediate test episodes. However, these non-convex strategies are eventually discarded as the MOQ-learning solution set gradually converges to extreme strategies, which are points (-19,124) and (-1,1) (Figure 6.3(a)). Quite the contrary, MOMCTS approaches discovers all strategies in the Pareto front, and keeps them in the search tree after they have been discovered. The weakness of MOMCTS-hv is that the longest decision sequences corresponding to the vectorial rewards (-17,74) and (-19,124) are more rarely discovered.

Stochastic setting Figure 6.4 shows the performance of MOMCTS-hv, MOMCTS-dom and MOQ-learning-m=21 with stochastic noise η ranging from 0.01 to 0.1. Comparatively to Figure 6.2, we can see that noise in the environment adversely affects the stability of all approaches, and particularly MOQ-learning. It affects MOMCTS-hv more than MOMCTS-dom in the low noise setting ($\eta = 0.01$). It affects comparably MOMCTS-hv and MOMCTS-dom in the high-noise setting.

In summary, the empirical validation on the artificial DST problem shows both the strengths and the weaknesses of MOMCTS approaches. On the positive side, MOMCTS



(a) $\eta = 0.01$



(b) $\eta = 0.1$

Figure 6.4: The hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning-m=21 versus training time in the stochastic environment with (a) $\eta = 0.01$ and (b) $\eta = 0.1$.

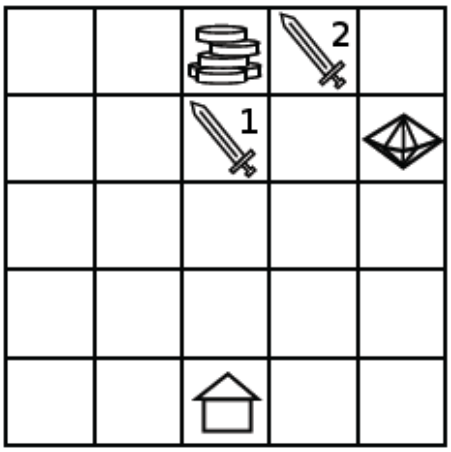


Figure 6.5: The Resource Gathering problem. The initial position of the agent is the mid-bottom case. Two resources (gold and gems) are located in fixed positions. Two enemy cases (marked by swords) send the agent back home with 10% probability.

approaches show able to find solutions lying in the non-convex regions of the Pareto front, as opposed to linear scalarization-based methods. Moreover, MOMCTS shows a relatively good robustness w.r.t. probabilistic transition model, comparatively to MOQ-learning. On the negative side, MOMCTS approaches are more computationally expensive than MOQ-learning (for 300,000 time steps, MOMCTS-hv takes 147 secs, MOMCTS-dom takes 49 secs versus 25 secs for MOQ-learning).

6.3 Resource Gathering problem

The MORL methods have been mostly applied to bi-objective SDM problems. For example, [Tesauro et al., 2007] optimizes both the performance and power consumption of a computing system. [Castelletti et al., 2011] balances between the benefit (the average performance) and risk (the worst-case performance) of a water reservoir control system.

As discussed in section 3.4, a challenge for MOO is to deal with many objectives ($d \geq 3$) due to the fact that, on one hand, the number of non-dominated solutions in the search space increases with the number d of objectives. On the other hand, the number of non-dominated solutions needed to approximate the entire Pareto front increases exponentially with d as well [4] [Ishibuchi et al., 2008].

In this section, we use a three-objective artificial problem – Resource Gathering to assess the scalability of MOMCTS approaches.

6.3.1 Problem statement

The Resource Gathering (RG) task first introduced in Barrett and Narayanan [2008] is carried out in a 5×5 grid (Figure 6.5). The action space of RG includes the same four actions (up, down, left and right) as in the DST problem. Starting from the home location, the goal of the agent is to gather two resources (gold and gems) and take them back home. Each time the agent reaches one resource location, the resource is picked up. Both resources can be carried by the agent at the same time. If the agent steps on one of the two enemy cases (indicated by swords), it may be attacked with 10% probability, in which case the agent loses all resources being carried and is returned to the home location immediately. The agent enters a terminal state when it returns home (including the case of being attacked) or when the time horizon $T = 100$ is reached. Five possible immediate reward vectors ordered as $(enemy, gold, gems)$ will be received upon the termination of a policy:

- $(-1, 0, 0)$ in case of an enemy attack;
- $(0, 1, 0)$ for returning home with only gold;
- $(0, 0, 1)$ for returning home with only gems;
- $(0, 1, 1)$ for returning home with both gold and gems;
- $(0, 0, 0)$ in all other cases.

The RG problem involves a discrete state space of 100 states corresponding to the 25 agent positions in the grid, multiplied by the four possible states of resources currently being held (none, gold only, gems only, both gold and gems). The vectorial reward associated to each policy π is calculated as follows:

Let $\mathbf{r} = (enemy, gold, gems)$ be the vectorial reward obtained by policy π after a L-step episode. The reward in each time step of the episode is noted by $\mathbf{r}_{\pi;L} = \mathbf{r}/L = (enemy/L, gold/L, gems/L)$. The policy is assessed by the average over 100 episodes of $\mathbf{r}_{\pi;L}$, where L is the length of each episode, favouring the discovering of gold and gems as soon as possible. Seven policies (Table 6.3 and Figure 6.6) corresponding to the non-dominated average vectorial rewards of the RG problem are identified by Vamplew et al. [2010]. The non-dominated vectorial rewards compose a convex Pareto front in the three dimensional space (Figure 6.7).

6.3.2 Experimental setting

In the RG problem, the MOMCTS approaches are assessed comparatively with the MOQ-learning algorithm, which independently optimizes the weighted sums of the three objective functions $(enemy, gold, gems)$ under m weight settings. In the three dimensional reward space, one weight setting is defined by a 2D vector (λ_i, λ_i'), with $\lambda_i, \lambda_i' \in [0, 1]$

[4]Furthermore, the increase of objective number makes it more difficult to visualize and assess the solution sets in the objective space.

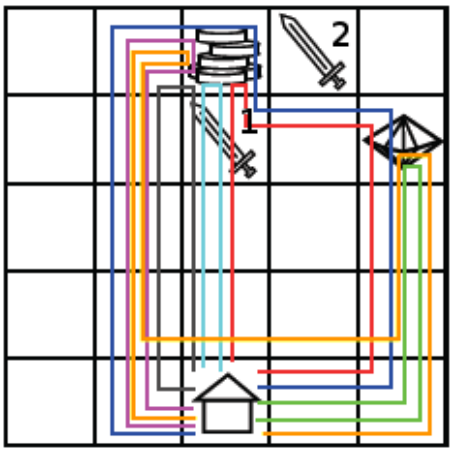


Figure 6.6: The seven policies in the Resource Gathering problem that correspond to the non-dominated vectorial rewards.

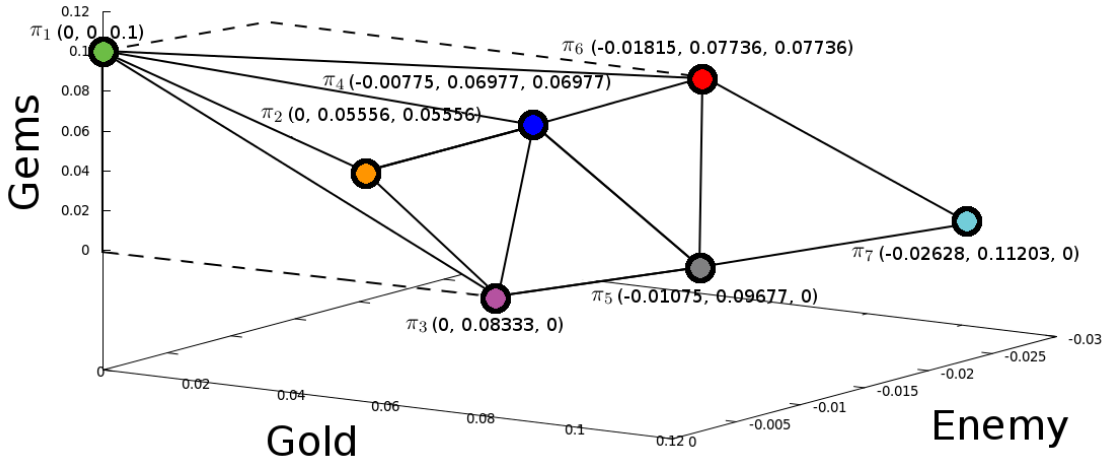


Figure 6.7: The seven non-dominated vectorial rewards in the Resource Gathering problem identified by Vamplew et al. [2010].

Table 6.3: The optimal policies for the Resource Gathering problem.

#	policy description	vectorial reward
π_1	Go directly to gems, avoiding enemies	(0,0,0.1)
π_2	Go to both gold and gems, avoiding enemies	$(0, 5.556 \times 10^{-2}, 5.556 \times 10^{-2})$
π_3	Go directly to gold, avoiding enemies	$(0, 8.333 \times 10^{-2}, 0)$
π_4	Go to both gold and gems, through enemy1 or enemy2 once	$(-7.75 \times 10^{-3}, 6.977 \times 10^{-2}, 6.977 \times 10^{-2})$
π_5	Go directly to gold, through enemy1 once	$(-1.075 \times 10^{-2}, 9.677 \times 10^{-2}, 0)$
π_6	Go to both gold and gems, through the enemies twice	$(-1.815 \times 10^{-2}, 7.736 \times 10^{-2}, 7.736 \times 10^{-2})$
π_7	Go directly to gold, through enemy1 twice	$(-2.628 \times 10^{-2}, 1.1203 \times 10^{-1}, 0)$

and $0 \leq \lambda_i + \lambda'_i \leq 1$. Let us denote the scalar rewards optimized by MOQ-learning as $r_i = (1 - \lambda_i - \lambda'_i) \times r_{enemy} + \lambda_i \times r_{gold} + \lambda'_i \times r_{gems}$, where l weights λ_i (respectively λ'_i) are evenly distributed in $[0, 1]$ for the gold (resp. gems) objective, subject to $\lambda_i + \lambda'_i \leq 1$, the total number of weight settings thus is $m = \frac{l(l-1)}{2}$.

The parameters of MOQ-learning and MOMCTS approaches have been selected after preliminary experiments, using the same amount of computational resources for a fair comparison.

For the MOQ-learning:

- The ϵ-greedy exploration is used with $\epsilon = 0.2$.
- Learning rate α is set to 0.2.
- The discount factor γ is set to 0.95.
- By taking $l = 4, 6, 10$, the number m of weight settings ranges in $\{6, 15, 45\}$.

The progressive widening parameter b in MOMCTS-hv is set to 2. The exploration vs exploitation (EvE) trade-off parameters associated to each objective are defined as $c_{enemy} = 1 \times 10^{-3}, c_{gold} = c_{gems} = 1 \times 10^{-4}$.

In MOMCTS-dom, the progressive widening parameter b is set to 1 (no progressive widening). The EvE trade-off parameter c_e is set to 0.1. The discount factor δ is set to 0.99.

The training time of all considered algorithms is 600,000 time steps (ca 17,200 tree-walks for MOMCTS-hv and 16,700 tree-walks for MOMCTS-dom. Like in the DST problem, the training process is equally divided into 150 phases. At the end of each training phase, the MOQ-learning and MOMCTS solution sets are tested in the RG simulator. Each solution (strategy) is launched 100 times and is associated the average vectorial reward (which might dominate the theoretical optimal ones due to the limited sample). The vectorial rewards of the solution set provided by each algorithm define its Pareto archive. The reference point z used in the hypervolume indicator calculation is set to

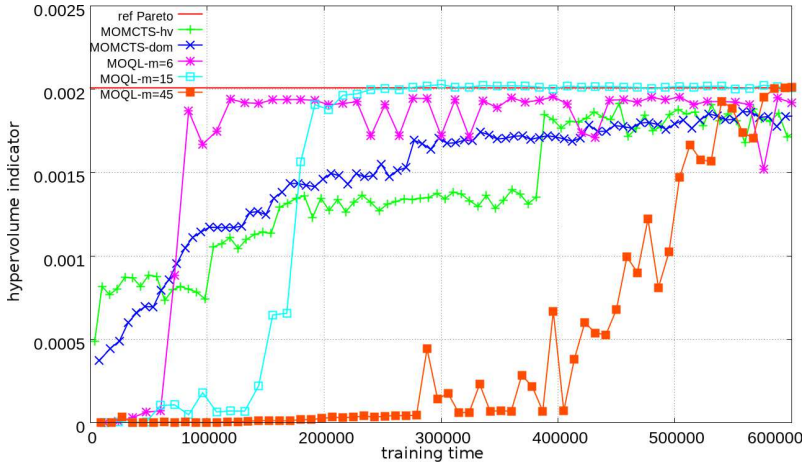


Figure 6.8: The Resource Gathering problem: Average hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning (with $m = 6$, 15 and 45) over 11 runs, versus number of time steps. The optimal hypervolume indicator 2.01×10^{-3} is indicated by the top line.

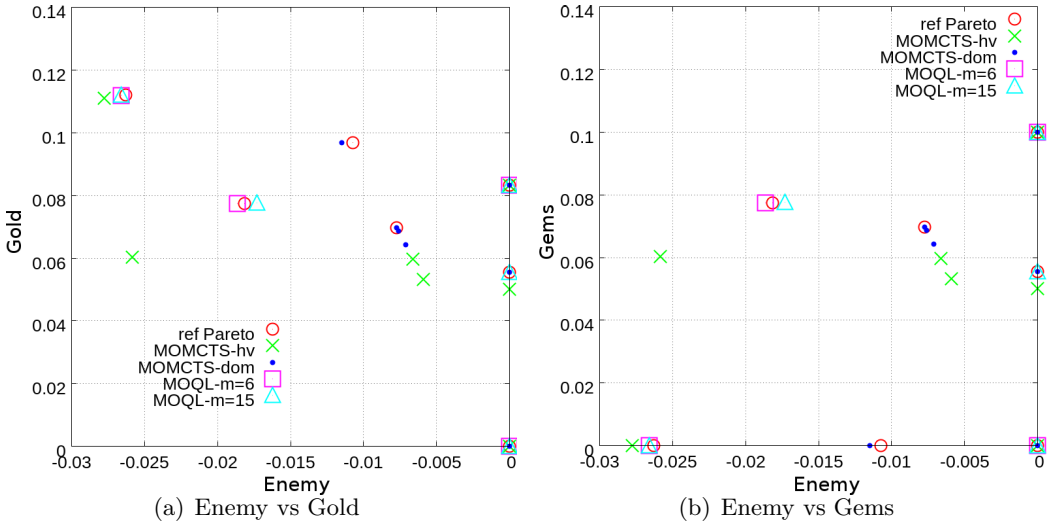
$(-0.33, -1 \times 10^{-3}, -1 \times 10^{-3})$, where -0.33 represents the maximum enemy penalty averaged in each time step of the episode, and the -1×10^{-3} values in the gold and gems objectives are taken to encourage the exploration of solutions with vectorial rewards lying in the hyper-planes $gold = 0$ and $gems = 0$. The optimal hypervolume indicator is 2.01×10^{-3}.

6.3.3 Results

Table 6.4: The Resource Gathering problem: Average hypervolume indicator of MOMCTS-hv, MOMCTS-dom and MOQ-learning (with $m = 6$, 15 and 45) over 11 runs. The optimal hypervolume indicator is 2.01×10^{-3}. Significantly better results are indicated in bold font (significance value $p < 0.05$ for the Student's t-test).

	HV($\times 10^{-3}$)		HV($\times 10^{-3}$)
MOMCTS-hv	1.735±0.304	MOMCTS-dom, $\delta = 0.9$	1.285±0.351
MOQ-learning, $m = 6$	1.933±0.04	MOMCTS-dom, $\delta = 0.98$	1.75±0.38
MOQ-learning, $m = 15$	**2.021±0.033**	MOMCTS-dom, $\delta = 0.99$	1.836±0.175
MOQ-learning, $m = 45$	**2.012±0.041**	MOMCTS-dom, $\delta = 0.999$	1.004±0.26

Table 6.4 shows the performance of MOMCTS-hv, MOMCTS-dom and MOQ-learning algorithms after 600,000 training times steps, measured by the hypervolume indicator. Figure 6.8 displays the evolution of hypervolume indicator in MOMCTS-hv, MOMCTS-dom and MOQ-learning with $m = 6, 15, 45$. The percentage of times out of 11 runs that each non-dominated vectorial reward is discovered for at least one test period during the training process of each algorithm is displayed in Figure 6.10. It is observed that with



(a) Enemy vs Gold (b) Enemy vs Gems

Figure 6.9: The vectorial rewards found by representative MOMCTS-hv, MOMCTS-dom and MOQ-learning with $m = 6, 15$ runs. Left: the points projected on the $Gems = 0$ plane. Right: the points projected on the $Gold = 0$ plane. The Pareto optimal points are marked by circles.

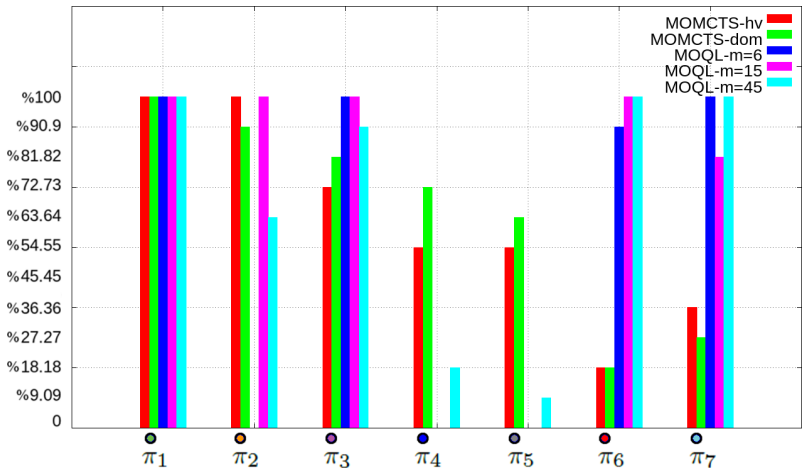


Figure 6.10: The percentage of of times out of 11 runs that each non-dominated vectorial reward was discovered by MOMCTS-hv,MOMCTS-dom and MOQ-learning with $m = 6, 15, 45$, during at least one test period.

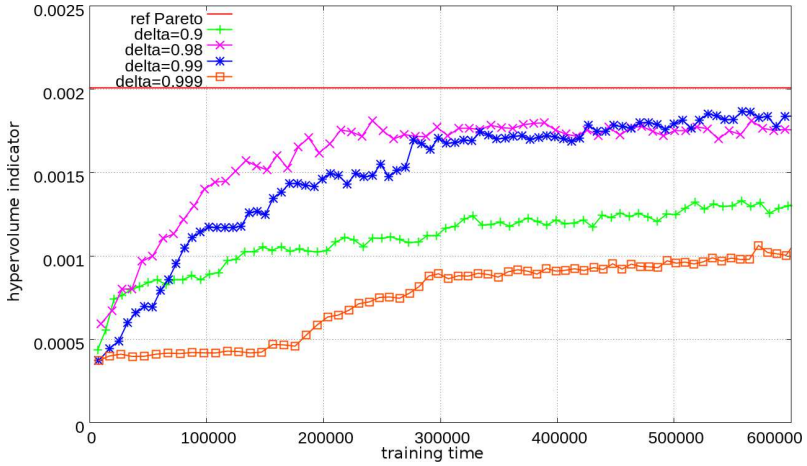


Figure 6.11: The hypervolume indicator performance of MOMCTS-dom with δ varying in $\{0.9, 0.98, 0.99, 0.999\}$, versus training time steps in the Resource Gathering problem.

$m = 6$ weight settings, the MOQ-learning performance stops improving after reaching a plateau of 1.9×10^{-3} at 120,000 time steps. Inspecting the Pareto archive, the difference between the performance plateau and the optimal performance (2.01×10^{-3}) is due to the non-discovery of policies π_2, π_4 and π_5 whose vectorial rewards are not covered by the 6 weight settings. MOQ-learning reaches the optimum when m increases (after 240,000 steps for $m = 15$ and 580,000 steps for $m = 45$).

The MOMCTS approaches are outperformed by MOQ-learning; their average hypervolume indicator reach 1.8×10^{-3} in the end of the training process, which is explained as the MOMCTS approaches rarely find the risky policies (π_6, π_7) (Figure 6.10). A tentative explanation for this fact is that risky non-dominated policies, such as π_6 and π_7, are hidden by dominated policies. For example, policy π_6 visits the enemy case twice; the neighbour nodes of this policy thus get the (-1,0,0) reward. As noted by [Coquelin and Munos, 2007], it may require an exponential time for the UCT algorithm to converge to the optimal node if this node is hidden by nodes with low reward.

As shown in Figure 6.11, the δ parameter governs the MOMCTS-dom performance. A low value ($\delta = 0.9$) leads to quickly forgetting the discovery of non-dominated rewards, turning MOMCTS-dom into pure exploration. Quite the contrary, high values of δ ($\delta = 0.999$) limit the exploration and likewise hinder the overall performance. The increasing interval between successive discoveries of non-dominated solutions (section 5.4.3) suggests that δ should be adjusted dynamically. This is a perspective for further work.

On the computational cost side, the average execution time of 600,000 training steps of in MOMCTS-hv, MOMCTS-dom and MOQ-learning are respectively 944 secs, 47 secs and 43 secs. Let us recall that the complexity of each tree-walk in MOMCTS-hv and MOMCTS-dom are respectively $\mathcal{O}(B|P|^{d/2}logN)$ and $\mathcal{O}(B \log N + d|P|)$, where B is average branching factor in the MOMCTS tree, P is Pareto archive and N is the number of tree-walks. As the size of the Pareto archive is close to 10, and the tree-depth is about

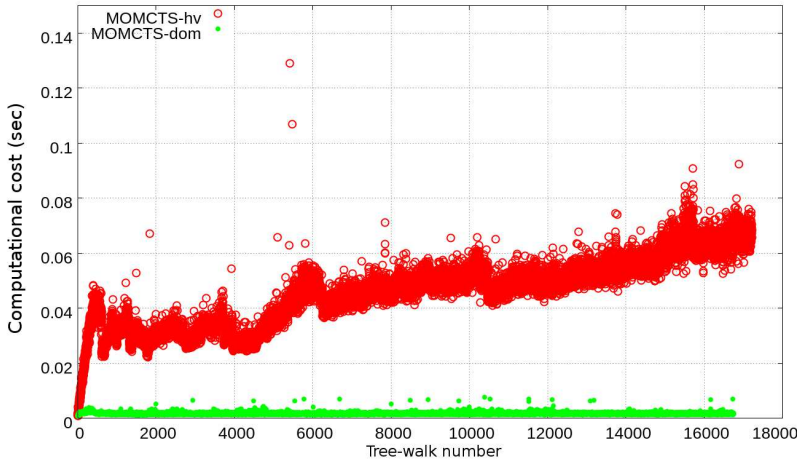


Figure 6.12: The Resource Gathering problem: average computational cost for one tree-walk for MOMCTS-hv and MOMCTS-dom over 11 independent runs. On average, each tree-walk in MOMCTS is ca. 35 training time steps.

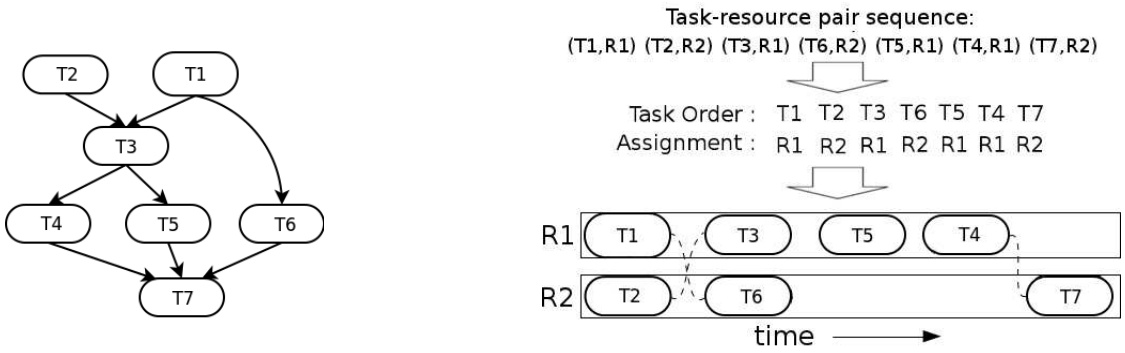
30 ($\log N \approx 30$) in most tree-walks of MOMCTS-hv and MOMCTS-dom, the fact that MOMCTS-hv algorithm is 20 times slower than MOMCTS-dom reflects their computational complexities.

As shown in Figure 6.12, the average cost of a tree-walk in MOMCTS-hv increases up to 20 times compared to that of MOMCTS-dom within the first 500 tree-walks, during which period the Pareto archive size $|P|$ grows. Afterwards, the cost of MOMCTS-hv gradually increases with the depth of the search tree ($\mathcal{O}(\log N)$). On the contrary, the computational cost of each tree-walk in MOMCTS-dom remains stable (between 1×10^{-3} secs and 2×10^{-3} secs) throughout the training process.

6.4 Grid Scheduling problem

Pertaining to the domain of autonomic computing [Tesauro et al., 2007; Perez, 2010], the problem of grid scheduling is concerned with scheduling the different tasks involved in the jobs on different computational resources. As tasks are interdependent and resources are heterogeneous, grid scheduling defines an NP-hard combinatorial optimization problem [Ullman, 1975]. We refer the reader to [Yu et al., 2008; Perez et al., 2010] for a comprehensive presentation of the field.

Grid scheduling naturally aims at minimizing the so-called makespan, that is the overall job completion time. But other objectives such as energy consumption, monetary cost, or the allocation fairness w.r.t. the resource providers become increasingly important. In the rest of this section, two objectives will be considered, the makespan and the cost of the solution. Due to its multi-objective nature, the grid scheduling problem has a set of Pareto optimal solutions available, among which only one will be executed in the reality. The final choice of execution plans is made by a coordinator (human or computer) whose decision



(a) (b)

Figure 6.13: Scheduling a job containing 7 interdependent tasks on a grid of 2 resources. Left: The dependency graph of tasks in the job. Right: The illustration of an execution plan.

depends on the observation of the entire Pareto optimal solution set. As the coordinator preference function is unknown, grid scheduling is a decision support application scenario of MOMCTS [Runarsson et al., 2012].

6.4.1 Problem statement

In grid scheduling, a job involves J tasks $T_1 \dots T_J$, partially ordered through a dependency relation; $T_i \to T_j$ denotes that task T_i must be executed before task T_j (Figure 6.13(a)). Each task T_i is associated with its unitary load L_i. Each task is assigned one out of M resources $R_1, \dots R_M$: resource R_k has computational efficiency $speed_k$ and unitary cost $cost_k$. Grid scheduling achieves the task-resource assignment and orders the tasks executed on each resource. A grid scheduling solution called execution plan is given as a sequence σ of (task-resource) pairs (Figure 6.13(b)).

Let $\rho(i) = k$ denote the index of the resource R_k on which T_i is executed. Let $\mathcal{B}(T_i)$ denote the set of tasks T_j which must either be executed before T_i ($T_j \to T_i$) or which are scheduled to take place before T_i on the same resource $R_{\rho(i)}$. The completion time of a task T_i is recursively computed as:

$$end(T_i) = \frac{L_i}{speed_{\rho(i)}} + max\{end(T_j), T_j \in \mathcal{B}(T_i)\}$$

where the first term is the time needed to process T_i on the assigned resource $R_{\rho(i)}$, and the second term expresses the fact that all jobs in $\mathcal{B}(T_i)$ must be completed prior to executing T_i.

Finally, grid scheduling is the two-objective optimization problem aimed at minimizing the overall scheduling makespan and cost:

$$\begin{array}{l}\text{Find } (\sigma) = argmin\ \{max\{end(T_j),\ j = 1 \dots J\}\ ; \\ \sum_{k=1\dots M} \frac{cost_k}{speed_k} \times \sum_{i\ s.t.\ \rho(i)=k} L_i\}\end{array}$$

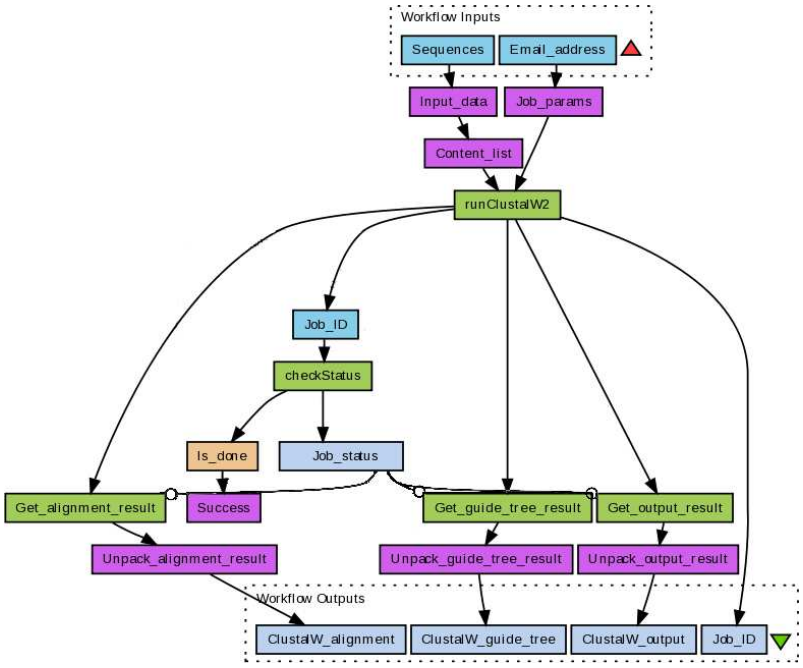


Figure 6.14: The EBI_ClustalW2 workflow.

6.4.2 Experimental setting

The state of the art in grid scheduling is achieved by stochastic optimization algorithms [Yu et al., 2008]. The two prominent multi-objective variants (NSGA-II [Deb et al., 2000] and SMS-EMOA [Beume et al., 2007], section 3.3.3) are therefore chosen as the baseline algorithms in our experiment.

A simulated grid environment containing 3 resources with different unit time costs and processing capabilities ($cost_1 = 20, speed_1 = 10; cost_2 = 2, speed_2 = 5; cost_3 = 1, speed_3 = 1$) is defined. We firstly compare the performance of MOMCTS approaches and baseline algorithms on a realistic bio-informatic workflow *EBI_ClustalW2* (Figure 6.14), which performs a ClustalW multiple sequence alignment using the EBI's WSClustalW2 service[5]. This workflow contains 21 tasks and 23 precedence pairs (graph density $q = 12\%$ [6]), assuming that all workloads are equal. Secondly, the scalability of MOMCTS approaches is tested through experiments based on artificially generated workflows containing respectively 20, 30 and 40 tasks with graph density $q = 15\%$.

As evidenced from the literature [Wang and Gelly, 2007], MCTS performances heavily depend on the so-called random phase (section 2.6.1). Preliminary experiments showed that a uniform action selection in the random phase was ineffective. A simple heuristic was thus used to devise a better suited action selection criterion in the random phase, as

[5]The complete description is available at *http://www.myexperiment.org/workflows/203.html*.

[6]The graph density q is defined as the portion of pairs (T_i, T_j) which are linked by a precedence constraint.

follows.

Let ECT_i define the expected completion time of task T_i (computed off-line, [Eswari and Nickolas, 2011]):

$$\mathrm{ECT}_i = L_i + max\{\mathrm{ECT}_j \ s.t. \ T_j \rightarrow T_i\}$$

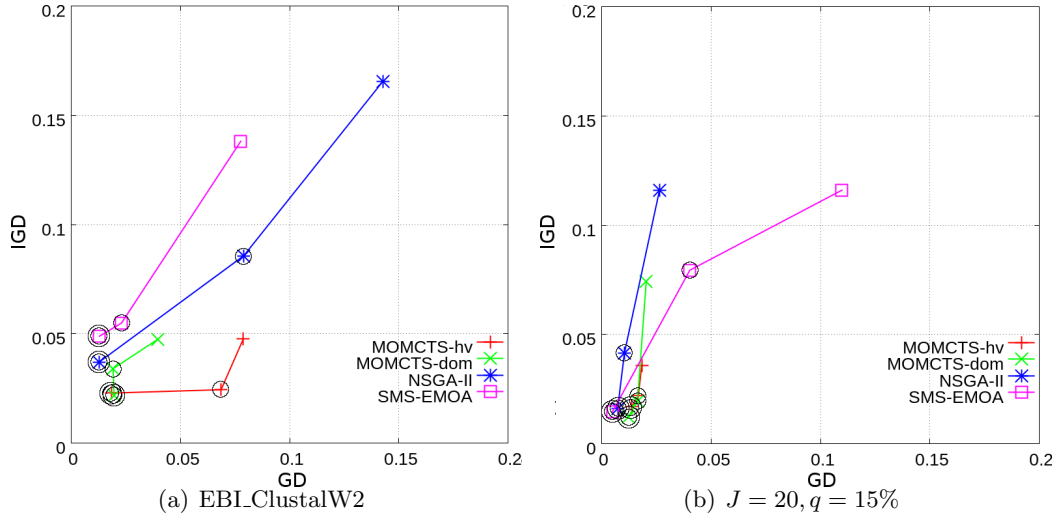
The heuristic action selection uniformly selects an admissible task T_i. It then compares ECT_i to all ECT_j for T_j admissible. If ECT_i is maximal, T_i is allocated to the resource which is due to be free at the earliest; if ECT_i is minimal, T_i is allocated to the resource which is due to be free at the latest. The random phase thus implements a default policy, randomly allocating tasks to resources, except for the most (respectively less) critical tasks that are scheduled with high (resp. low) priority.

The parameters of all algorithms have been selected after preliminary experiments, using the same amount of computational resources for a fair comparison. The progressive widening parameter b is set to 2 in both MOMCTS-hv and MOMCTS-dom. In MOMCTS-hv, the exploration vs. exploitation (EvE) trade-off parameters associated to the makespan and cost objectives, c_{time} and c_{cost} are both set to 5×10^{-3}. In MOMCTS-dom, the EvE trade-off parameters c_e is set to 1, and the discount factor δ is set to 0.99. The parameters used for NSGA-II (respectively SMS-EMOA) involve a population size of 200 (resp. 120) individuals, of which 100 are selected and undergo stochastic unary and binary variations (resp. one-point re-ordering, and resource exchange among two individuals). For all three algorithms, the number N of tree-walks a.k.a. evaluation budget is set to 10,000. The reference point in each experiment is set to (z_t, z_c), where z_t and z_c respectively denote the maximal makespan and cost.

Due to the fact that the true Pareto front in the considered problems is unknown, as said, we use a reference Pareto front P^* gathering all non-dominated vectorial rewards obtained in all runs of all three algorithms in lieu of the true Pareto front. The performance indicators are defined by the generational distance (GD) and inverted generational distance (IGD) (section 3.3.3) between the actual Pareto front P found in the run and the reference Pareto front P^*. In the grid scheduling experiment, the IGD indicator measures the diversity of solutions in the Pareto front P, like the hypervolume indicator does in DST.

6.4.3 Results

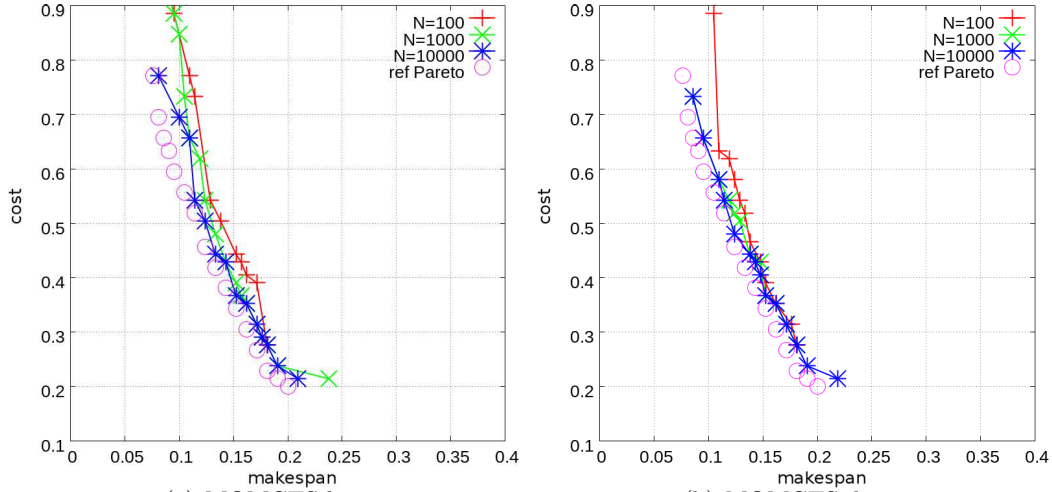
Figure 6.15 displays the GD and IGD of MOMCTS-hv, MOMCTS-dom, NSGA-II and SMS-EMOA on EBI_ClustalW2 workflow scheduling and on artificial jobs with a number J of tasks ranging in $20, 30$ and 40 with graph density $q = 15\%$. Figure 6.16 shows the Pareto front discovered by MOMCTS-hv, MOMCTS-dom, NSGA-II and SMS-EMOA on the EBI_ClustalW2 workflow after $N = 100,\ 1000$ and 10000 policy evaluations (tree-walks), comparatively to the reference Pareto front. In all considered problems, the MOMCTS approaches are outperformed by the baselines in terms of the GD indicator. While they quickly find good solutions, they fail to discover the reference Pareto front. In the meanwhile, they yield a better IGD performance than the baselines, indicating that on



(a) EBI_ClustalW2 (b) $J = 20, q = 15\%$

(c) $J = 30, q = 15\%$ (d) $J = 40, q = 15\%$

Figure 6.15: The generational distance (GD) and inverted generational distance (IGD) for $N = 100,\ 1000$ and 10000 of MOMCTS-hv, MOMCTS-dom, NSGA-II and SMS-EMOA on (a): EBI_ClustalW2; (b)(c)(d): artificial problems with number of tasks J and graph density q. Each performance point after 1000 and 10 000 evaluations are respectively marked by single and double circles.



(a) MOMCTS-hv (b) MOMCTS-dom

(c) NSGA-II (d) SMS-EMOA

Figure 6.16: Progression of the Pareto-optimal solutions found for $N = 100$, 1000 and 10000 for MOMCTS-hv, MOMCTS-dom, NSGA-II and SMS-EMOA on the EBI_ClustalW2 workflow. The reference Pareto front is indicated by circles.

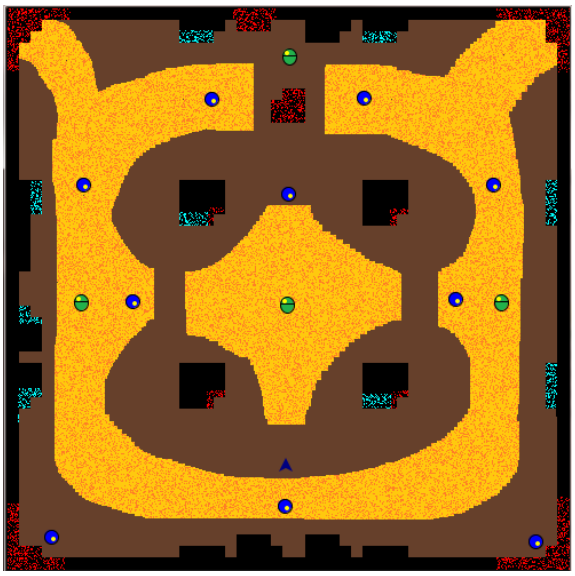


Figure 6.17: The PTSP map.

average a single run of MOMCTS approaches spreads to a wider region in the objective space, and reaches a better approximation of the true Pareto front.

Overall, the main weakness of MOMCTS approaches is their computational runtime. The computational cost of MOMCTS-hv and MOMCTS-dom are respectively 5 and 2.5 times higher than that of NSGA-II and SMS-EMOA[7]. This is indeed a serious problem for real-time decision settings. However, in many real-world problems, the evaluation cost dominates by several orders of magnitude the search cost, which alleviates this weakness of MOMCTS.

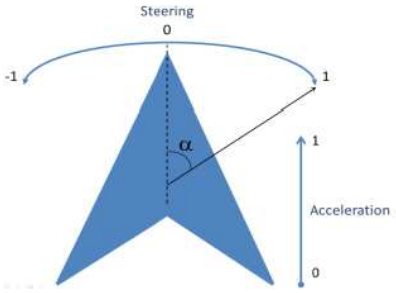
6.5 Physical Travelling Salesman Problem

6.5.1 Problem statement

The physical travelling salesman problem (PTSP) extends the travelling salesman problem (TSP) to the problem of robot navigation. The TSP is a well known combinatorial optimization problem where a series of cities (or nodes) and the cost of travelling between them are known. A salesman must visit all cities exactly once and go back to the starting city by following the path of minimum cost.

In PTSP, the agent (i.e. the salesman) governs a ship that must visit a series of way-points scattered in a map (Figure 6.17) as quickly as possible. Beside the goal of minimizing the time to visit all way-points, the agent must also consider two supplementary goals – minimize the fuel consumption and the damage caused by passing through

[7]On workflow EBI_ClustalW2, the average execution time of MOMCTS-hv, MOMCTS-dom, NSGA-II and SMS-EMOA are respectively 142 secs, 74 secs, 31 secs and 32 secs.



Action ID	Acceleration	Steering
0	No (0)	No (0)
1	No (0)	Left (-1)
2	No (0)	Right (1)
3	Yes (1)	No (0)
4	Yes (1)	Left (-1)
5	Yes (1)	Right (1)

Figure 6.18: Action space of PTSP.

dangerous areas (lava surface) or when confronting obstacles during the trip.

The 6 actions in PTSP (Figure 6.18) are determined by two different inputs : acceleration and steering. Acceleration can take two possible values (on and off), while steering can turn the ship to the left, right or keep it straight. Each acceleration action consumes one unit of fuel.

The state of the ship is described as by a 3-tuple (o_t, v_t, p_t), including the orientation o_t, velocity v_t and position p_t vectors. A known deterministic transition model is used to modify the state of PTSP according to the executed actions.

The orientation of the ship is changed as shown in Eq.(6.1), given the ship's orientation o_t in the last time step and the rotation angle α caused by the steering action. Eq.(6.2) indicates how the velocity vector is modified, given the previous velocity v_t, the new orientation d_{t+1}, an acceleration constant K, and a frictional loss factor L. In this case, the acceleration input determines the value of T_t : set to 1 if the action implies acceleration or 0 otherwise. Finally, Eq.(6.3) updates the position of the ship by adding the velocity vector to its location p_t in the previous time step. The inertia of the ship is kept in the velocity vector v_t, which makes the task of navigating the ship more challenging.

$$o_{t+1} \leftarrow \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix} \cdot o_t \tag{6.1}$$

$$v_{t+1} \leftarrow (v_t + (o_{t+1} T_t K)) \cdot L \tag{6.2}$$

$$p_{t+1} \leftarrow p_t + v_{t+1} \tag{6.3}$$

Different elements in PTSP map are shown in Figure 6.19, among which both the obstacles and the lava surface in the map may damage the ship. One unit of damage is taken by the ship for every time step it spends on the lava surface. A collision on the normal obstacle does a low damage (15 units) to the ship and produces an elastic collision, which modifies the velocity of the ship (both in direction and module). Collisions on the damaging surface bring a high damage (30 units) to the ship. Elastic surface does not damage the ship, but produces an elastic collision.

6.5.2 Problem analysis

Robot navigation in PTSP faces two main challenges. Firstly, the policy space in PTSP is large (B^T, with $B = 6$ the branching factor and $T = 1000$ the minimal time horizon

	Normal ground surface: This is a normal, non-damaging surface.
	Lava surface:: 1 unit of damage is taken by the ship for every game cycle it spends in this surface.
	Normal obstacle: This is an obstacle that does **low** damage to the ship.
	Damaging surface: This is an obstacle that does **high** damage to the ship.
	Elastic surface: This obstacle does **not** damage the ship, but the ship will bounce when colliding.
	The player / ship: This is the ship driven by the player.
	Waypoint: One of the waypoints to be collected.
	Fuel Tank: This canister provides fuel to the ship. It is **not** mandatory to pick them up.

Figure 6.19: Legends of elements in an PTSP map.

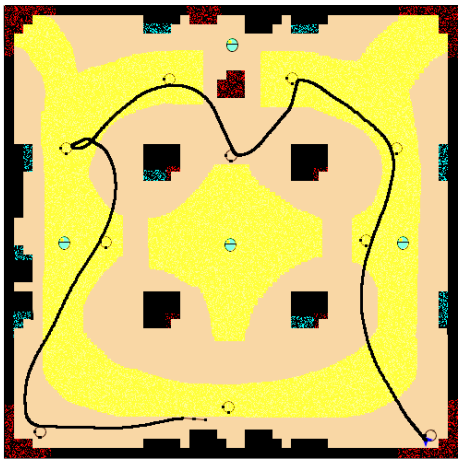


Figure 6.20: An example path which traverse all way-points in the PTSP map. 1530 actions are made to correct the direction and accelerate the ship in this path.

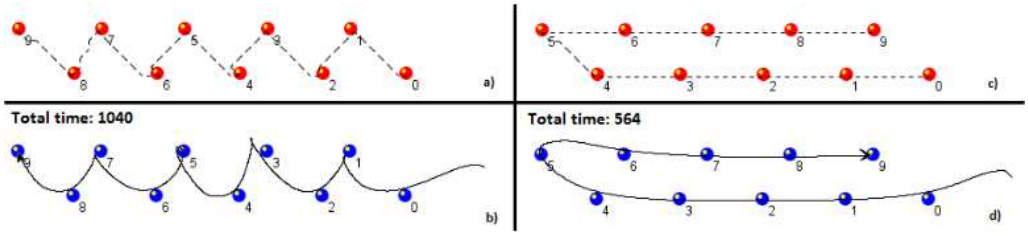


Figure 6.21: Paths with different visit orders corresponds to different length in PTSP extracted from [Powley et al., 2012]. Figures *b* and *d* respectively shows the trace of the ship by following orders defined by Figure *a* and *c*. Note that the time is not simply determined by the travel length due to the inertia effect.

which enables a tour which successfully visits all way-points (Figure 6.20)).

Secondly, the PTSP imposes real-time planning requirements to the controller. In real-world robot navigation problems, robot decisions about the next movement usually need to be made within milliseconds. Specifically, in the 2013 CIG PTSP competition, 1000 ms is given for the initialization of controllers, and the planning interval between two consecutive actions is 40 ms. Under the real-time constraint, the on-line planning ability becomes essential in the controller design.

In order to overcome the challenges in the above two aspects, we firstly fix the visit order of all way-points in PTSP, and the task of MOMCTS is to achieve the local navigation from one way-point to another one by following the fixed order.

Secondly, we introduce the persistence parameter M, imposing that each selected action out of 6 is executed repetitively along M consecutive steps. The fine-grained adjustment of M is a critical issue as it controls the size of the local navigation problem, of which the time horizon is divided by M. On the other hand, it restricts the flexibility of the navigation. Preliminary experiments were used to adjust M, a range from 5 to 30 is considered in the experiments.

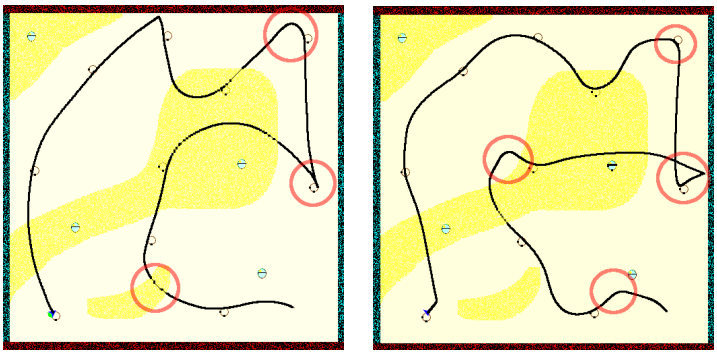
The third point is to incorporate the varying preference application scenario described in section 4.5.

Techniques used in the design of MOMCTS-based PTSP controller are formally presented as the following.

6.5.2.1 Problem decomposition

Taking inspiration from Powley et al. [2012], the PTSP problem is decomposed into two sub-problems with different levels of granularity. Firstly, at a global level, a macro-planner defines the order in which the way-points are visited. Secondly, at a local level, a steering controller based on the MOMCTS algorithm determines how to go from the current position to the next way-point.

Noticing that different visit orders of way-points produce paths with varying lengths (Figure 6.21), the macro-planner achieves an approximated optimal order of way-points through the resolution of a regular TSP problem, using an A^* algorithm for an affordable



(a) $M = 10$ (b) $M = 30$

Figure 6.22: Examples of the path followed by the MOMCTS controller. Left: The path created with $M = 10$, in which rotations are smooth. Right: The path created with $M = 30$, in which most turns are in 90° angle.

approximation [8].

The MOMCTS is only in charge of steering the ship from the current position to the next way-point in plan.

The benefit of the problem decomposition is to reduce the length of the sequential decision making in PTSP from 1000 to ca 100 time steps.

6.5.2.2 Macro actions

Following again Powley et al. [2012], a hyper-parameter M is introduced, controlling the persistence of actions in the PTSP controller. Specifically, each action is repeated for M time steps. The M value influences the performance of steering controller as follows.

On one hand, smaller M value corresponds to a control with higher flexibility. Suppose that one steering step corresponds to a rotation of 3°. Then setting $M = 30$ restricts the ship to only making 90° turns. However, algorithms using this setting will only find paths that have to bounce off walls or follow convoluted routes to line up with way-points. A choice of $M = 10$ corresponds to 30° turns, which allows for a finer control of the ship (Figure 6.22).

On the other hand, greater M values increase the controller's ability to plan ahead. Assuming that given the same amount of computation time, the forward planning ability of the model exponentially increases with M.

In summary, M controls the trade-off between the flexibility and forward planning ability of the controller, which needs to be tuned in practical applications.

[8] Once the map is given, the point to point distances are estimated by the scanline floodfill algorithm [Lieberman, 1978].

6.5.2.3 Varying preference modes

Complying with the time constraint of 40 ms, the action selection in the MOMCTS based steering controller is achieved as follows. Firstly, 7 auxiliary objectives are defined and computed for each tree-walk:

$$\mathbf{r} = (r_{dist}, r_{distNext}, r_{leftWaypointNb}, r_{time}, \\ r_{fuel}, r_{damage}, r_{leftFueltankNb})$$

where r_{dist} is the instant distance to the next way-point in plan, $r_{distNext}$ is the instant distance to the point after the next one. The two terms $(r_{dist}, r_{distNext})$ are designed to guide the steering controller towards the next way-point, while achieving a favourable position from which to set off towards the second planned way-point. The third objective $r_{leftWaypointNb}$ is the number of way-points to be visited in the remainder of the trajectory. The 4th term r_{time} records the minimum time spent before visiting the next way-point computed by the simulator. The fifth and sixth term (r_{fuel}, r_{damage}) counts the fuel consumption and damage caused by the steering policy. The last objective $r_{leftFueltankNb}$ is the number of fuel tanks left to be collected. All objectives are required to be minimized.

In each decision step, MOMCTS is launched, estimating the Pareto front w.r.t. the 7 objectives. The choice of actions among Pareto optimal solutions is determined by following the varying preference application scenario (section 4.5). Specifically, three types of situations are considered: the set-off mode, the high-speed mode and normal mode. Each situation is associated a weight setting, reflecting the prior knowledge of steering in different environments. The action minimizing the weighted sum (Eq. (4.6)) of the 7 objectives is retained [9].

Set-off mode: In the situation where the fuel storage is affluent and the speed of the ship is small (like at the moment when the ship sets off), the ship is encouraged to accelerate to reach a normal speed as soon as possible. The following weight setting is therefore taken, which ignores the cost in fuel consumption and discourages collecting fuel tanks:

$$\mathbf{w} = (w_{dist} = 10, w_{distNext} = 3, w_{leftWaypointNb} = 500, w_{time} = 10, \\ \mathbf{w_{fuel}} = \mathbf{0}, w_{damage} = 1, \mathbf{w_{leftFueltankNb}} = \mathbf{100})$$

High speed mode: When the speed of the ship overpasses a threshold ($|v_t| \geq 1.2$ in the PTSP environment), the risk of collision and target missing increases drastically. The acceleration (fuel consumption) is discouraged in this situation and the following weight setting is taken :

$$\mathbf{w} = (w_{dist} = 10, w_{distNext} = 3, w_{leftWaypointNb} = 500, w_{time} = 10, \\ \mathbf{w_{fuel}} = \mathbf{5}, w_{damage} = 1, w_{leftFueltankNb} = 500)$$

[9] Non-linear preference functions can also be used in the selection among Pareto optimal solutions. The design and learning of user preference functions is an increasingly active field in RL [Brochu et al., 2007; Fürnkranz et al., 2012; Akrour et al., 2012]. Developing population based preference function also composes one of the major perspectives of our work. More discussions on this perspective will be found in section 6.6.

Normal mode: In all other cases, the following weight setting keeps a balance between objectives:

$$\mathbf{w} = (w_{dist} = 10, w_{distNext} = 3, w_{leftWaypointNb} = 500, w_{time} = 10,$$
$$w_{fuel} = 1, w_{damage} = 1, w_{leftFueltankNb} = 500)$$

In summary, the calculation of multiple Pareto optimal solutions in MOMCTS allows the use of multiple preference modes which enables the integration of prior knowledge in the controller and allows for a better control in face of the dynamics in PTSP problem. The MOMCTS controller thus treats the varying preference application scenario of multiple-policy MORL. Interestingly, while the optimization of the weighted sum of objectives can be solved by single-objective optimizers, the experiments show that searching for the set of Pareto optimal solutions in the planning phase does enforce a better exploration of the multi-dimensional objective space and eventually bring a better solution in the execution phase (more in section 6.5.5).

6.5.3 Baseline algorithms

In the PTSP problem, MOMCTS is compared with its ancestor versions – the MCTS and MC (Monte-Carlo) algorithms. MCTS is the original version of MOMCTS which solves single-objective SDM problems (section 2.6.1). In our experiment, MCTS solves the multi-objective PTSP problem by optimizing the upper bound of weighted sum of the 7-objective vectorial reward in PTSP. Let $\hat{\mathbf{r}}$ denote the average rewards cumulated in the 7 auxiliary objectives of PTSP, and let $\mathbf{w}$ denote the associated weight setting. The node selection criterion in MCTS is defined as

$$g_{\mathbf{w}}(s,a) = \hat{r}_{s,a} + \sqrt{c_e \ln(n_s)/n_{s,a}} \tag{6.4}$$

where $\hat{r}_{s,a} = \hat{\mathbf{r}} \cdot \mathbf{w}$. For a fair comparison, as the situation/mode of the ship is known, the weights associated to the objectives in MCTS take the same value as in MOMCTS.

MC method optimizes the same value function $g_{\mathbf{w}}(s,a)$ as in MCTS. However, unlike MCTS which chooses actions in multiple levels of the search tree, the MC method chooses the action maximizing $g_{\mathbf{w}}(s,a)$ among the direct descendants of the root node, and thereafter randomly chooses actions until reaching the time horizon of the local search problem. By limiting the search tree depth to 1, MCTS degrades to the MC.

6.5.4 Experimental setting

Goals of experiments Experiments on PTSP are carried out with three goals in mind. The first goal is to compare the performance of MOMCTS, MCTS and MC. The second goal is to assess the scalability of algorithms depending on the complexity of the map. The third goal is to examine the sensitivity of algorithms w.r.t. the hyper-parameter M.

The presented experiments rely on the PTSP framework used in 2013 CIG PTSP competition. In each PTSP game, the initial 1000 ms are used to execute the macro-planner. The planning interval between two consecutive actions is 40 ms. The rotation

step α is fixed to 3°. The friction factor L is fixed to 0.99. The acceleration constant K is fixed to 0.025. Three maps of different complexities (simple, medium, difficult) are used in our experiment, each of which contains 10 way-points (Figure 6.23). The game is terminated in four cases: if the ship has not visited all 10 way-points within 5000 time steps, or more than 800 time steps are spent without reaching a new way-point, or the ship runs out of fuel, or the cumulative damage suffered by the ship is more than 5000.

As preliminary experiments show, the MOMCTS-hv algorithm fails to find solutions in PTSP problem under the real-time constraint due to its excessive computational cost. The results reported in our experiment are therefore based on the MOMCTS-dom (noted by MOMCTS in the following), MCTS, and MC algorithm.

The MOMCTS, MCTS and MC based steering controllers implement the same 6 macro actions as in [Powley et al., 2012], where the 6 original actions are executed repetitively for a fixed number M of times. After preliminary experiments, the time horizon T in all tested algorithms is fixed to 5, which correspond to a search space of $6^5 = 46,656$ possible policies. Note that only a negligible fraction of the policy space can be evaluated under the time constraint (ca 1000 policies).

After tuning the parameters in MOMCTS, MCTS and MC, the search tree depth D in MOMCTS, MCTS and MC is respectively set to 3, 3 and 1 [10]. In MOMCTS, the EvE trade-off parameter $c_e = 10$, and the discount factor δ is set to 0.5. The progressive widening parameter b is set to 2 in both MOMCTS and MCTS. c_e is set to 100 in both MCTS and MC.

All algorithms are executed for 11 times in each map, and 11 result vectors $\mathbf{R} = (R_{time}, R_{damage}, R_{fuel})$, representing the overall completion time, damage and fuel consumption of the traversal trip, are returned by the PTSP engine. As each run yields a result vector, the result vectors generated by 11 runs are associated with a hypervolume indicator, which measures the performance of each algorithm, with the reference point z set to $(5000, 5000, 5000)$.

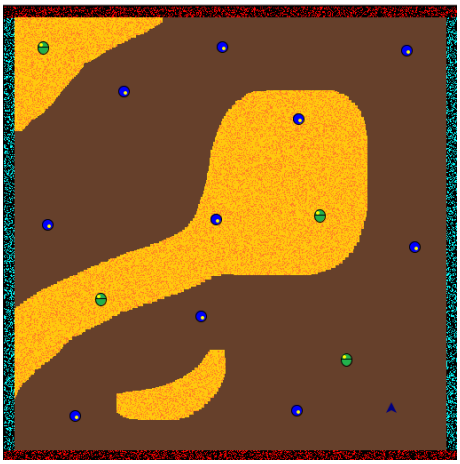
6.5.5 Results

Table 6.5: The best hypervolume indicator of the set of result vectors $\mathbf{R} = (R_{time}, R_{damage}, R_{fuel})$ obtained by MOMCTS, MCTS and MC over 11 runs (each run generates one single result vector $\mathbf{R}$) in the three test maps, with the reference point z set to $(5000, 5000, 5000)$. The best results are indicated in bold font.

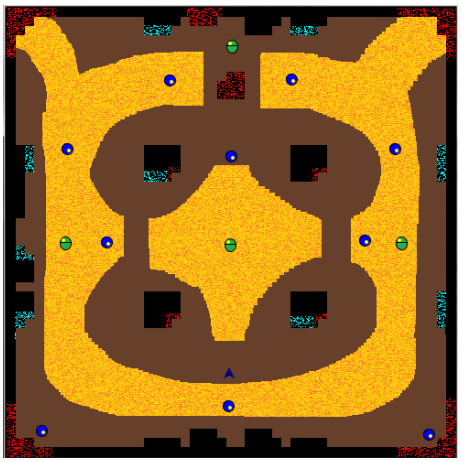
	MOMCTS	MCTS	MC
Simple map ($\times 10^{10}$)	**8.24**	7.57	7.87
Medium map ($\times 10^{10}$)	**7.45**	6.82	7.25
Difficult map ($\times 10^{10}$)	4.28	3.71	**4.39**

As the macro action length M influences the performance of steering controllers in

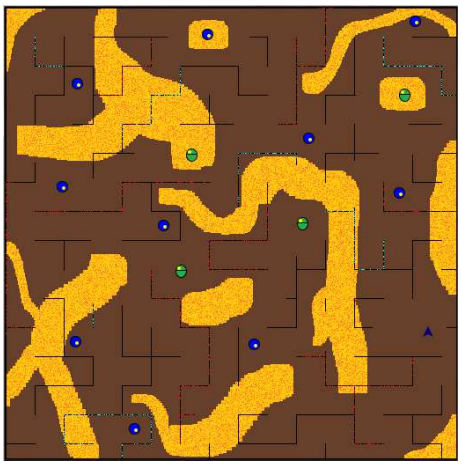
[10] Correspondingly, the length of the random phase in MOMCTS, MCTS and MC equalling $T - D$ are respectively 2, 2 and 4.



(a) Simple Map

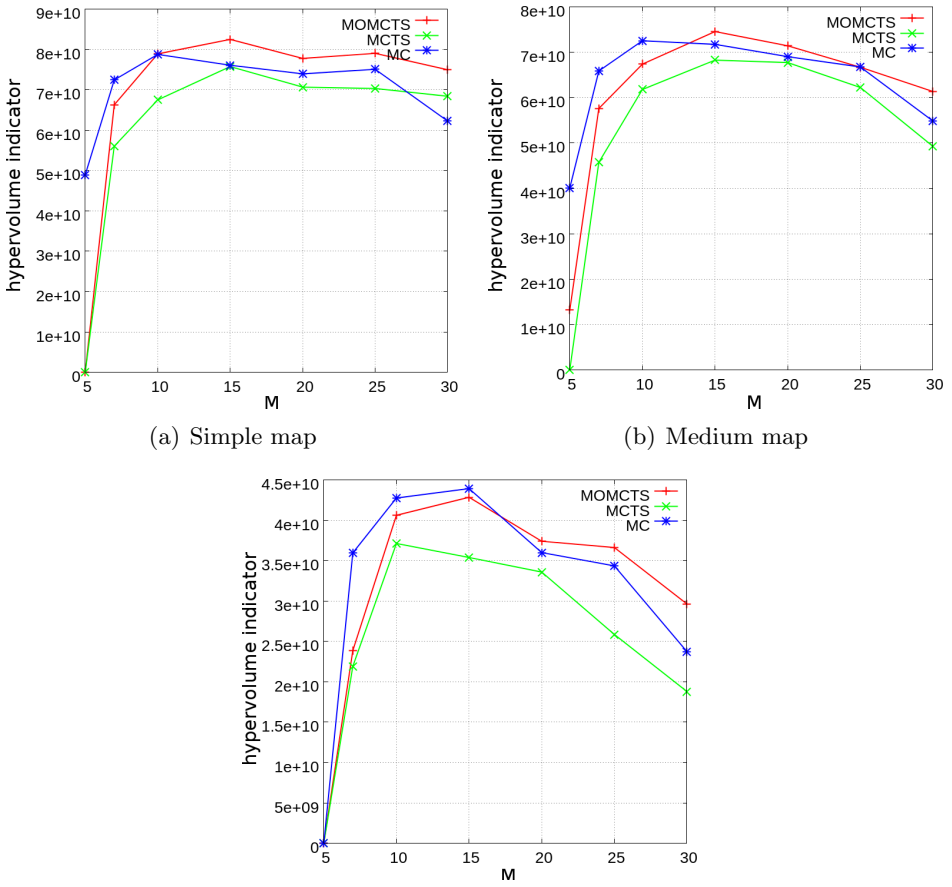


(b) Medium Map



(c) Difficult Map

Figure 6.23: Three representative maps extracted from the 2013 CIG PTSP competition toolkit. All three maps are of the same size 512 pixel × 512 pixel. Blue points are way-points to be visited, and green points are fuel tanks. The difference between these maps lies in the obstacle setting. The simple map (a) does not contain any obstacle, while the medium map (b) contains more rugged walls and 4 obstacles in the middle. The difficult map (c) is a maze-like arena (black segments indicate walls) in which no straight path exist between any two way-points.



(a) Simple map

(b) Medium map

(c) Difficult map

Figure 6.24: Sensitivity analysis: impact of macro-action length M on hypervolume indicator performance of MOMCTS, MCTS and MC in simple, medium and difficult maps.

PTSP problem, experiments with M varying in $\{1, 5, 7, 10, 15, 20, 25, 30\}$ have been carried out to assess such influence, and the results are shown in Figure 6.24. It is observed that, the best runs of MOMCTS, MCTS and MC all correspond to the M value between 10 and 15, which generates 30° to 45° of rotation or 10 to 20 pixels of displacement (assuming that the average speed of the ship is around 1.2) in each macro-action. Knowing that the time horizon T is fixed to 5 in our experiment, $M \in [10, 15]$ then corresponds to an activity region with the maximal radius of 100 pixels, which is in the same scale of the average distance between two neighbour way-points (between 150 and 200) in the considered maps. Inspecting the traversal trace of the ship in different maps, we find out that greater M values ($M \geq 20$) decrease the algorithm performance due to the reduced control flexibility which result in more target missing and bounces on the walls. The harmful effect of large M values is even more obvious in more difficult maps (Figure 6.24(b),(c)).

Table 6.5 further shows the comparison among the hypervolume indicator of solution sets obtained by MOMCTS,MCTS and MC with different M values. MOMCTS outperforms MCTS and MC in both the simple and medium map, while MC outperforms MOMCTS for the difficult map. The optimal solution sets are displayed in Figure 6.25. We find that most MCTS solutions are dominated by the MOMCTS and MC ones. Recall that the only difference between MC and MCTS is that MC has a smaller search tree depth and longer random phase, a tentative explanation for this phenomenon is that more samples are need to assess the value of a node.

When comparing MOMCTS and MC, it is observed that MC solutions are mostly better than MOMCTS rewards in the *time* objective, and are worse in the *damage* and *fuel* objectives. This is probably due to the single-policy nature of MC methods. By inspecting the reward log of optimization process, we notice that in most linearly scalarized rewards in MC (whose amplitude reaches 10,000), the impact of *fuel* and *damage* objectives (r_{fuel}, r_{damage} and $r_{fuelTank}$) is less than 10%, which easily gets ignored when compared with the *time* objective related rewards. Further inspections show that discarding the damage and fuel related rewards is actually beneficial to MC in the difficult map : the difficult map requires the ship to frequently change its orientation through accelerations (fuel consumption) or bounces against walls (damage). Compared to MOMCTS, MC tends to sacrifice fuel and damage related objectives to achieve better time related performance. Such behaviour is particularly effective in the difficult map.

Our tentative explanation for the phenomenon that MOMCTS is dominated by MC in the difficult map is due to the fact that the weights is biased towards the simple and medium map. The analysis of the champion of 2013 CIG PTSP competition, which is an MCTS based controller, shows that the authors have been optimizing the weight settings using the CMA-ES [Hansen, 2006] algorithm. Further experiments will likewise use MO-CMA-ES to refine the weight vectors used in MOMCTS [11]. With the manual weight vectors, the MOMCTS-based PTSP controller got the 2nd rank in the multi-objective PTSP competition out of 8 competitors, and the 4th place in the single-objective PTSP

[11] In order to get better results in PTSP, another possibility is to refine the preference mode design in MOMCTS-based controller. As longer term perspectives, non-linear scalarization function in the PTSP framework will also be considered.

competition out of 30 competitors [12].

Through the PTSP problem, the ability of MOMCTS to exploit prior knowledge has been demonstrated. On the negative side, it has been shown that the actual results depend on the accuracy of some prior knowledge, here, the weight vectors.

6.6 Partial conclusion

The careful experimental study of MOMCTS on artificial and real-world like problems has shown the potential and current limitations of the approach.

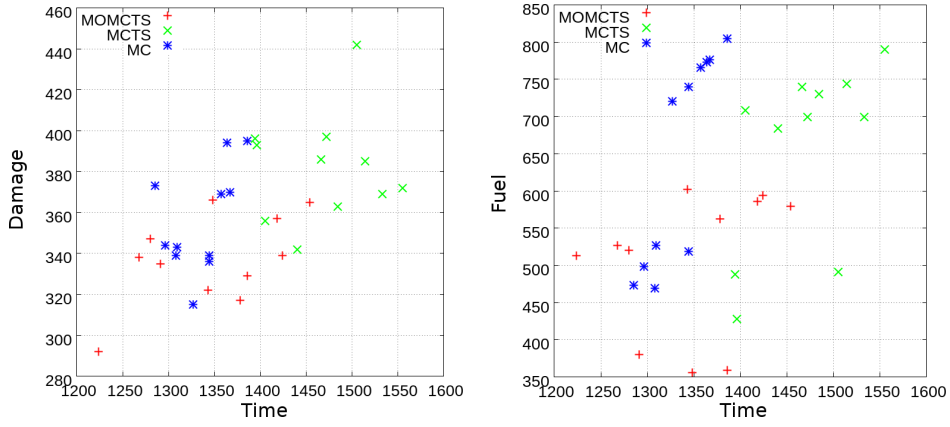
On the positive side, MOMCTS does not suffer from intrinsic limitations w.r.t. non-convex Pareto front. In the meanwhile, it requires quite some tuning efforts to best fit the problem at hand. These efforts include 1) the design of a heuristic roll-out policy to be used in the random phase, as shown for the grid scheduling problem; 2) the adjustment of the depth vs breadth of the search tree, specifically between the time horizon and forward exploration ability, as shown in the PTSP experiments.

The comparison of the MOMCTS-hv and MOMCTS-dom shows that some progress can be done regarding the trade-off between the diversity of the discovered Pareto front and the computation time. Specifically, MOMCTS-dom is almost at side w.r.t. computational cost regarding the state of the art, while MOMCTS-hv is slower by at least one order of magnitude.

The good robustness w.r.t. probabilistic transition model of both MOMCTS-hv and MOMCTS-dom has been demonstrated.

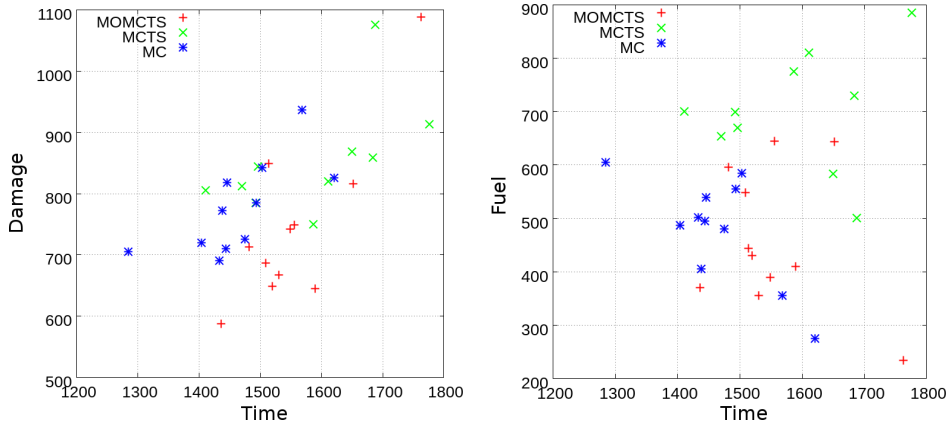
A next perspective for further extension of MOMCTS thus regards how to enforce the discovery of the whole range of Pareto front in the MOMCTS-dom. Further work will be considered with optimization of the prior in both Grid Scheduling and PTSP as well.

[12] In the single-objective PTSP competition, only the maximization of the number of way-points visited under a fixed time budget is considered. The results of both competitions are available on the site *http://www.ptsp-game.net/bot_mo_rankings.php*.



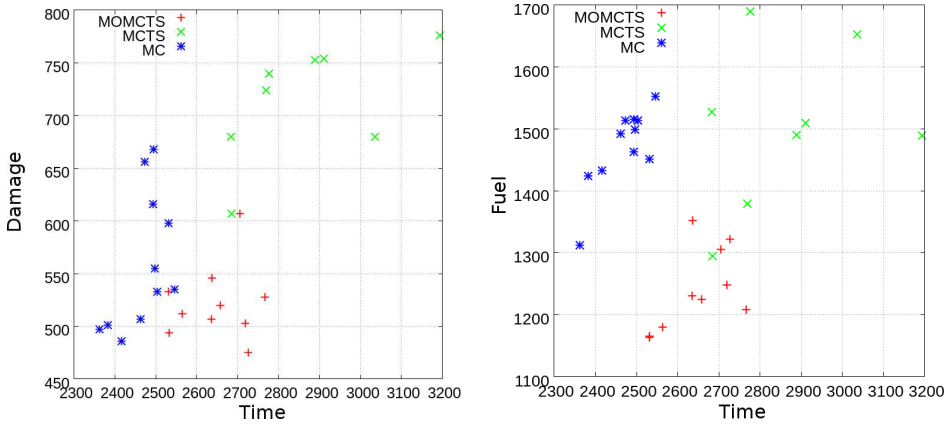
(a) Simple map: Time vs Damage

(b) Simple map: Time vs Fuel



(c) Medium map: Time vs Damage

(d) Medium map: Time vs Fuel



(e) Difficult map: Time vs Damage

(f) Difficult map: Time vs Fuel

Figure 6.25: The result vector $(R_{time}, R_{damage}, R_{fuel})$ distribution of MOMCTS, MCTS and MC in the three tested maps.

Part IV

Conclusion and Perspectives

Chapter 7

Conclusions

This thesis investigates the multi-objective sequential decision making (MOSDM) problem. Besides the known difficulties of sequential decision making (size of the search space, delayed effects of decisions, possibly stochastic transition models), the MOSDM faces the main difficulty at the core of multi-objective optimization setting, namely the lack of total order among solutions, here policies.

7.1 Summary of contributions

Our contribution is to extend the MCTS framework [Kocsis and Szepesvári, 2006] to multi-objective sequential decision making. Inheriting the scalable advantage of MCTS in single-objective SDM, the proposed MOMCTS framework handles the multi-objective SDM problem by searching for several Pareto optimal solutions within a single tree. The main challenge in this work is to extend the node selection rule in MCTS to the multi-objective case. This was done by using, besides the estimation of the upper bound of rewards associated to each node, an archive P of all solutions discovered during the search in terms of the Pareto front in the objective space. By maintaining this archive, auxiliary performance indicators can be computed for each tree-walk.

Inspired from the MOO literature, the first performance indicator is the hypervolume indicator measuring how a given non-dominated solution extends and improves the Pareto front. A main merit of the hypervolume indicator is to enforce the diversity of the discovered solution, favoring the sampling of the Pareto front. A difficulty of the hypervolume indicator is that it is not often operational as the majority of tree-walk solutions are dominated. Therefore, a penalty over the hypervolume indicator is imposed by considering the distance of the dominated solutions to the envelope of the current Pareto front. The weakness of the hypervolume indicator based performance measure is two-fold. On the one hand, it is computationally expensive. Experimentally it was not usable in the PTSP experiments in Chapter 6. The second weakness is that the hypervolume indicator is not invariant w.r.t. the monotonous transformation of the objectives.

Addressing these limitations, the second performance indicator is proposed, where the binary reward associated to each tree-walk is 1 iff this tree-walk is non dominated w.r.t. the current Pareto archive. The dominance reward based performance indicator enjoys two advantages. Firstly, its computational complexity is linear w.r.t. the number of objectives. Secondly, it is invariant under the monotonous transformation of the objective functions. However, as there is only a tiny percent of tree-walks that find non-dominated solutions, the average dominance reward in most nodes are negligible in front of the exploration term, making the MOMCTS degenerate to pure random search. In order to overcome this

limitation, a cumulative discounted (CD) updating procedure is used to update the value of nodes based on the dominance reward. Therefore, the second performance indicator is called cumulative discounted dominance (CDD) reward.

These approaches have been validated on both artificial (Deep Sea Treasure and Resource Gathering) and real-world (Grid Scheduling and Physical Travelling Salesman Problem) problems. The experimental results on the Deep Sea Treasure problem confirm a main merit of the proposed approaches, their ability to discover policies lying in the non-convex regions of the Pareto front. To our knowledge [1], this feature is unique in the MORL literature. The experiments on Resource Gathering show that MOMCTS-dom enjoys a better scalability than MOMCTS-hv because of the linear computational cost of Pareto dominance test w.r.t. the number of objectives. Such scalability of MOMCTS-dom is further confirmed by the Physical Travelling Salesman Problem experiments, in which 7 objectives are optimized in an on-line manner.

In the counterpart, MOMCTS approaches suffer from two main weaknesses. Firstly, as shown on the Grid Scheduling and Physical Traveling Salesman Problem, some domain knowledge is required to enforce the exploration efficiency of MOMCTS. Secondly, as evidenced in Resource Gathering problem, the presented approaches hardly discover "risky" policies which lie in an unpromising region (the proverbial needle in the haystack).

In summary, this work can be seen as a proof of concept of the application of MOMCTS framework for the MOO setting. The promising result is that the presented work reaches a decent performance, despite the fact that they are less mature than the approaches in the RL field.

7.2 Future directions

This work opens theoretical and applicative perspectives for further studies.

The main theoretical perspective concerns the properties of the cumulative discounted (CD) reward updating mechanism in the general (single-objective) dynamic optimization context. Besides, the consistency analysis of the current node selection criteria (including hypervolume indicator and dominance reward) is required to provide a guideline for the future reward design within the MOMCTS framework.

On the applicative side, firstly, the linear scalarization preference function used in the Physical Traveling Salesman Problem experiment can be extended to more general (for example – non-linear) forms, which may allow more natural and interactive user preference expressions.

Another most interesting algorithmic perspective regards the adjustment of the CD

[1] A general polynomial result of MOO has been proposed by Chatterjee [2007], which claims that for all irreducible MDP with multiple long-run average objectives, the Pareto front can be ϵ-approximated in time polynomial in ϵ. However this claim relies on the assumption that *finding some Pareto optimal point can be reduced to optimizing a single objective: optimize a convex combination of objectives using as set of positive weights* (page 2, Chatterjee [2007]), which does not hold for non-convex Pareto fronts. Furthermore, the approach relies on the ϵ-approximation of the Pareto front proposed by Papadimitriou and Yannakakis [2000], which assumes the existence of an oracle telling for each vectorial reward whether it is ϵ-Pareto-dominated (Thm. 2, page 4, Papadimitriou and Yannakakis [2000]).

reward updating mechanism (Eq.(5.7)). As said, the discovery of non-dominated solutions is increasingly more rare along the search; the adjustment of the δ parameter should compensate for this effect. An option would be to consider the discovery of new non-dominated solutions along the extreme-value theory setting [De Haan and Ferreira, 2007], and to adjust δ accordingly.

Bibliography

Adibi, M., Zandieh, M., and Amiri, M. (2010). Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems with Applications*, 37(1):282–287. (Cited on pages ii and 7)

Akrour, R., Schoenauer, M., and Sebag, M. (2012). April: active preference learning-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 116–131. Springer. (Cited on pages 67 and 110)

Aliprantis, C. D. and Chakrabarti, S. K. (2000). *Games and decision making*. Oxford University Press New York. (Cited on pages i and 3)

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256. (Cited on pages 24 and 27)

Auer, P., Ortner, R., and Szepesvári, C. (2007). Improved rates for the stochastic continuum-armed bandit problem. In *Learning Theory*, pages 454–468. Springer. (Cited on page 32)

Auger, A., Bader, J., Brockhoff, D., and Zitzler, E. (2009). Theory of the hypervolume indicator: optimal μ-distributions and the choice of the reference point. In *FOGA'09*, pages 87–102. ACM. (Cited on pages vi, 48, and 76)

Auger, D. (2011). Multiple tree for partially observable monte-carlo tree search. In *Applications of Evolutionary Computation*, pages 53–62. Springer. (Cited on page 32)

Auger, D., Couetoux, A., and Teytaud, O. (2013). Continuous upper confidence trees with polynomial exploration–consistency. In *Machine Learning and Knowledge Discovery in Databases*, pages 194–209. Springer. (Cited on page 32)

Back, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *Evolutionary computation, IEEE Transactions on*, 1(1):3–17. (Cited on page 44)

Bader, J. and Zitzler, E. (2011). Hype: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary Computation*, 19(1):45–76. (Cited on pages 52 and 53)

Barrett, L. and Narayanan, S. (2008). Learning all optimal policies with multiple criteria. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *ICML'08*, pages 41–47. ACM. (Cited on pages vii, 8, 60, 61, 63, and 93)

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition. (Cited on page 52)

Bellman, R. (1986). Dynamic programming and lagrange multipliers. *The Bellman Continuum: A Collection of the Works of Richard E. Bellman*, page 49. (Cited on pages 15 and 17)

Berthier, V., Doghmen, H., and Teytaud, O. (2010). Consistency modifications for automatically tuned Monte-Carlo Tree Search. In Blum, C. and Battiti, R., editors, *LION4*, pages 111–124. LNCS 6073, Springer-Verlag. (Cited on pages iv, 28, and 72)

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont. (Cited on page 12)

Bertsekas, D. P. and Tsitsiklis, J. N. (1995). Neuro-dynamic programming: An overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE. (Cited on pages xvii, 18, 19, 20, and 21)

Beume, N., Fonseca, C. M., Lopez-Ibanez, M., Paquete, L., and Vahrenhold, J. (2009). On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5):1075–1082. (Cited on pages 52 and 76)

Beume, N., Naujoks, B., and Emmerich, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653 – 1669. (Cited on pages 43, 49, 52, 76, and 101)

Beyer, H.-G. and Sendhoff, B. (2007). Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33):3190–3218. (Cited on page 52)

Bourki, A., Chaslot, G., Coulm, M., Danjean, V., Doghmen, H., Hoock, J.-B., Hérault, T., Rimmel, A., Teytaud, F., Teytaud, O., et al. (2011). Scalability and parallelization of monte-carlo tree search. In *Computers and Games*, pages 48–58. Springer. (Cited on pages 32 and 33)

Bowman Jr, V. J. (1976). On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In *Multiple criteria decision making*, pages 76–86. Springer. (Cited on page 58)

Brochu, E., De Freitas, N., and Ghosh, A. (2007). Active preference learning with discrete choice data. In *NIPS*. (Cited on page 110)

Brucker, P. and Brucker, P. (2007). *Scheduling algorithms*, volume 3. Springer. (Cited on page 3)

Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory*, pages 23–37. Springer. (Cited on page 26)

Castelletti, A., Corani, G., Rizzolli, A., Soncinie-Sessa, R., and Weber, E. (2002). Reinforcement learning in the operational management of a water system. In *IFAC Workshop*

on Modeling and Control in Environmental Issues, Keio University, Yokohama, Japan, pages 325–330. (Cited on pages ii, vi, 7, and 59)

Castelletti, A., Pianosi, F., and Restelli, M. (2011). Multi-objective fitted q-iteration: Pareto frontier approximation in one single run. In *ICNSC*, pages 260–265. (Cited on pages 66, 67, and 92)

Castelletti, A., Pianosi, F., and Soncini-Sessa, R. (2008). Receding horizon control for water resources management. *Applied Mathematics and Computation*, 204(2):621–631. (Cited on pages 66 and 67)

Cazenave, T. (2006). A phantom-go program. In *Advances in Computer Games*, pages 120–125. Springer. (Cited on page 31)

Charnes, A. and Cooper, W. W. (1957). Management models and industrial applications of linear programming. *Management Science*, 4(1):38–91. (Cited on page 43)

Chaslot, G., Bakkes, S., Szita, I., and Spronck, P. (2008a). Monte-carlo tree search: A new framework for game ai. In *AIIDE*. (Cited on page 28)

Chaslot, G., Chatriot, L., Fiter, C., Gelly, S., Hoock, J.-B., Perez, J., Rimmel, A., and Teytaud, O. (2008b). Combining expert, offline, transient and online knowledge in monte-carlo exploration. (Cited on page 76)

Chatterjee, K. (2007). Markov decision processes with multiple long-run average objectives. *FSTTCS 2007 Foundations of Software Technology and Theoretical Computer Science*, 4855:473–484. (Cited on pages 60 and 122)

Childs, B. E., Brodeur, J. H., and Kocsis, L. (2008). Transpositions and move groups in monte carlo tree search. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 389–395. IEEE. (Cited on page 31)

Ciancarini, P. and Favini, G. P. (2009). Monte-Carlo Tree Search techniques in the game of kriegspiel. In Boutilier, C., editor, *IJCAI'09*, pages 474–479. (Cited on pages ii, 6, 28, and 31)

Comsa, I. S., Aydin, M., Zhang, S., Kuonen, P., and Wagen, J.-F. (2012). Multi objective resource scheduling in lte networks using reinforcement learning. *International Journal of Distributed Systems and Technologies (IJDST)*, 3(2):39–57. (Cited on pages 66 and 67)

Coquelin, P. and Munos, R. (2007). Bandit algorithms for tree search. *arXiv preprint cs/0703062*. (Cited on page 98)

Couëtoux, A., Doghmen, H., and Teytaud, O. (2012). Improving the exploration in upper confidence trees. In *Learning and Intelligent Optimization*, pages 366–371. Springer. (Cited on page 32)

Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., and Bonnard, N. (2011). Continuous upper confidence trees. In *Learning and Intelligent Optimization*, pages 433–445. Springer. (Cited on page 32)

Couetoux, A., Milone, M., Brendel, M., Doghmen, H., Sebag, M., Teytaud, O., et al. (2011). Continuous rapid action value estimates. In *The 3rd Asian Conference on Machine Learning (ACML2011)*, volume 20, pages 19–31. (Cited on page 32)

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo Tree Search. In *Proc. Computers and Games*, pages 72–83. (Cited on pages 28 and 31)

De Haan, L. and Ferreira, A. (2007). *Extreme value theory: an introduction*. Springer. (Cited on pages viii and 123)

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, pages 55–58. Chichester. (Cited on pages 35, 37, and 41)

Deb, K. and Gupta, H. (2006). Introducing robustness in multi-objective optimization. *Evolutionary Computation*, 14(4):463–494. (Cited on page 52)

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In Schoenauer, M. et al., editor, *PPSN VI*, pages 849–858. LNCS 1917, Springer Verlag. (Cited on pages 36, 38, 43, 45, 49, and 101)

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2002). Scalable multi-objective optimization test problems. In *Proceedings of the Congress on Evolutionary Computation (CEC-2002),(Honolulu, USA)*, pages 825–830. Proceedings of the Congress on Evolutionary Computation (CEC-2002),(Honolulu, USA). (Cited on page 86)

Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2009). Engineering route planning algorithms. In *Algorithmics of large and complex networks*, pages 117–139. Springer. (Cited on page 14)

Edelsbrunner, H. and Shah, N. R. (1996). Incremental topological flipping works for regular triangulations. *Algorithmica*, 15(3):223–241. (Cited on page 74)

Eswari, R. and Nickolas, S. (2011). Expected completion time based scheduling algorithm for heterogeneous processors. *Information Communication and Management–International Proceedings of Computer Science and Information Technology*. (Cited on page 102)

Fikes, R. E. and Nilsson, N. J. (1972). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3):189–208. (Cited on page 14)

Finck, S., Hansen, N., Ros, R., and Auger, A. (2010). Real-parameter black-box optimization benchmarking 2010: Presentation of the noisy functions. Technical report, Citeseer. (Cited on page 52)

Fleischer, M. (2003). The measure of Pareto optima. applications to multi-objective metaheuristics. In *EMO'03*, pages 519–533. LNCS 2632, Springer Verlag. (Cited on pages vi and 48)

Fogel, D. B. (2006). *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. John Wiley & Sons. (Cited on page 24)

Fogel, L., Owens, A., and Walsh, M. (1966). Artificial intelligence through simulated evolution. (Cited on page 44)

Fürnkranz, J., Hüllermeier, E., Cheng, W., and Park, S.-H. (2012). Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine learning*, 89(1-2):123–156. (Cited on pages 67 and 110)

Gábor, Z., Kalmár, Z., and Szepesvári, C. (1998). Multi-criteria reinforcement learning. In *ICML'98*, pages 197–205. Morgan Kaufmann. (Cited on pages vi, 7, 55, 59, and 63)

Gelly, S., Hoock, J.-B., Rimmel, A., Teytaud, O., Kalemkarian, Y., et al. (2008). On the parallelization of monte-carlo planning. In *ICINCO*. (Cited on page 33)

Gelly, S. and Silver, D. (2007). Combining online and offline knowledge in UCT. In Ghahramani, Z., editor, *ICML'07*, pages 273–280. ACM. (Cited on pages ii, 7, 28, 29, and 31)

Goh, C. K. and Tan, K. C. (2007). An investigation on noisy environments in evolutionary multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 11(3):354–381. (Cited on page 52)

Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99. (Cited on pages 44 and 45)

Goldfeld, S. M., Quandt, R. E., and Trotter, H. F. (1966). Maximization by quadratic hill-climbing. *Econometrica: Journal of the Econometric Society*, pages 541–551. (Cited on page 14)

Hansen, N. (2006). The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer. (Cited on pages vi, 7, 76, and 115)

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107. (Cited on page 14)

Heidrich-Meisner, V. and Igel, C. (2009). Hoeffding and bernstein races for selecting policies in evolutionary direct policy search. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 401–408. ACM. (Cited on pages 24 and 52)

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366. (Cited on page 24)

Houli, D., Zhiheng, L., and Yi, Z. (2010). Multiobjective reinforcement learning for traffic signal control using vehicular ad hoc network. *EURASIP Journal on Advances in Signal Processing*, 2010:7. (Cited on pages 66 and 67)

Hsu, F.-H. (2002). *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press. (Cited on page 3)

Ishibuchi, H., Tsukamoto, N., and Nojima, Y. (2008). Evolutionary many-objective optimization: A short review. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2419–2426. IEEE. (Cited on pages 53, 57, and 92)

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *arXiv preprint cs/9605103*. (Cited on page 13)

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34(5-6):975–986. (Cited on page 24)

Knowles, J., Thiele, L., and Zitzler, E. (2006). A tutorial on the performance assessment of stochastic multiobjective optimizers. *Tik report*, 214:327–332. (Cited on page 37)

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In Fürnkranz, J., Scheffer, T., and Spiliopoulou, M., editors, *ECML'06*, pages 282–293. Springer Verlag. (Cited on pages i, 3, 28, 29, and 121)

Kolesnikov, A. (2003). *Efficient algorithms for vectorization and polygonal approximation*. University of Joensuu. (Cited on page 74)

Lazzerini, B., Marcelloni, F., and Vecchio, M. (2010). A multi-objective evolutionary approach to image quality/compression trade-off in jpeg baseline algorithm. *Applied Soft Computing*, 10(2):548–561. (Cited on page 52)

Lieberman, H. (1978). How to color in a coloring book. *ACM SIGGRAPH Computer Graphics*, 12(3):111–116. (Cited on page 109)

Lin, J. G. (1976). Three methods for determining pareto-optimal solutions of multiple-objective problems. In *Directions in large-scale systems*, pages 117–138. Springer. (Cited on page 41)

Littman, M. L. (1996). *Algorithms for sequential decision making*. PhD thesis, Brown University. (Cited on page 14)

Liu, W., Tan, Y., and Qiu, Q. (2010). Enhanced q-learning algorithm for dynamic power management with performance constraint. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 602–605. European Design and Automation Association. (Cited on pages 66 and 67)

Lizotte, D. J., Bowling, M., and Murphy, S. A. (2012). Linear fitted-q iteration with multiple reward functions. *Journal of Machine Learning Research*, 13:3253–3295. (Cited on pages 60, 61, 63, 66, and 67)

Loshchilov, I. (2013). *Surrogate-Assisted Evolutionary Algorithms*. PhD thesis, Université Paris Sud-Paris XI. (Cited on pages 24, 45, and 53)

Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial intelligence*, 55(2):311–365. (Cited on pages i and 3)

Mannor, S. and Shimkin, N. (2004). A geometric approach to multi-criterion reinforcement learning. *Journal of Machine Learning Research*, pages 325–360. (Cited on pages vi, 7, and 59)

Mansley, C. R., Weinstein, A., and Littman, M. L. (2011). Sample-based planning for continuous action markov decision processes. In *ICAPS*. (Cited on page 32)

Maravall, D. and de Lope, J. (2002). A reinforcement learning method for dynamic obstacle avoidance in robotic mechanisms. *Computational Intelligent Systems. World Scientific, Singapore*, pages 485–494. (Cited on pages 66 and 67)

Marglin, S. A. (1967). Public investment criteria; benefit-cost analysis for planned economic growth,. (Cited on page 41)

Marler, R. T. and Arora, J. S. (2010). The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 41(6):853–862. (Cited on pages 38 and 41)

Meisner, E. M., Adviser-Isler, V., and Adviser-Trinkle, J. (2009). Learning controllers for human-robot interaction. (Cited on pages 66 and 67)

Meyer, J. (1987). Two-moment decision models and expected utility maximization. *The American Economic Review*, pages 421–430. (Cited on page 5)

Nakhost, H. and Müller, M. (2009). Monte-Carlo exploration for deterministic planning. In Boutilier, C., editor, *IJCAI'09*, pages 1766–1771. (Cited on pages ii, 7, and 28)

Natarajan, S. and Tadepalli, P. (2005). Dynamic preferences in multi-criteria reinforcement learning. In *ICML'05*. ACM. (Cited on pages vi, 7, 59, 60, and 61)

Palmers, P., McConnaghy, T., Steyaert, M., and Gielen, G. (2009). Massively multi-topology sizing of analog integrated circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 706–711. European Design and Automation Association. (Cited on page 52)

Papadimitriou, C. H. and Yannakakis, M. (2000). On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92. IEEE Computer Society. (Cited on page 122)

Pareto, V. (1896). *Cours d'economie politique*. Librairie Droz. (Cited on page 35)

Perez, J. (2010). *Apprentissage artificiel pour l'ordonnancement des tâches dans les grilles de calcul*. PhD thesis, PhD thesis. (Cited on page 99)

Perez, J., Germain-Renaud, C., Kégl, B., and Loomis, C. (2009). Responsive elastic computing. In *Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, pages 55–64. ACM. (Cited on pages 66 and 67)

Perez, J., Germain-Renaud, C., Kégl, B., and Loomis, C. (2010). Multi-objective reinforcement learning for responsive grids. *Journal of Grid Computing*, 8(3):473–492. (Cited on page 99)

Perny, P. and Weng, P. (2010). On finding compromise solutions in multiobjective markov decision processes. In *ECAI*, pages 969–970. (Cited on page 59)

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697. (Cited on page 13)

Ponsen, M. J., Gerritsen, G., and Chaslot, G. (2010). Integrating opponent models with monte-carlo tree search in poker. In *Interactive Decision Theory and Game Theory*. (Cited on page 31)

Powley, E. J., Whitehouse, D., and Cowling, P. I. (2012). Monte carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 234–241. IEEE. (Cited on pages vii, xxi, 8, 31, 108, 109, and 112)

Purshouse, R. C. and Fleming, P. J. (2007). On the evolutionary optimization of many conflicting objectives. *Evolutionary Computation, IEEE Transactions on*, 11(6):770–784. (Cited on page 53)

Rechenberg, I. (1978). *Evolutionsstrategien*. Springer. (Cited on page 44)

Robbins, H. (1985). Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, pages 169–177. Springer. (Cited on pages ii, 6, and 25)

Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113. (Cited on pages xix, 55, and 65)

Rubinstein, R. Y. and Kroese, D. P. (2004). *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer. (Cited on page 24)

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering. (Cited on page 22)

Runarsson, T. P., Schoenauer, M., and Sebag, M. (2012). Pilot, rollout and monte carlo tree search methods for job shop scheduling. In *Learning and Intelligent Optimization*, pages 160–174. Springer. (Cited on page 100)

Saadatseresht, M., Mansourian, A., and Taleai, M. (2009). Evacuation planning using multiobjective evolutionary optimization approach. *European Journal of Operational Research*, 198(1):305–314. (Cited on page 52)

Saravanan, R., Ramabalan, S., Ebenezer, N., and Dharmaraja, C. (2009). Evolutionary multi criteria design optimization of robot grippers. *Applied Soft Computing*, 9(1):159–172. (Cited on page 52)

Schaffer, J. D. (1985). Some experiments in machine learning using vector evaluated genetic algorithms. Technical report, Vanderbilt Univ., Nashville, TN (USA). (Cited on page 45)

Shabani, N. (2009). *Incorporating flood control rule curves of the Columbia river hydroelectric system in a multireservoir reinforcement learning optimization model*. PhD thesis, University of British Columbia. (Cited on pages 66 and 67)

Shin, S.-Y., Lee, I.-H., Kim, D., and Zhang, B.-T. (2005). Multiobjective evolutionary optimization of dna sequences for reliable dna computing. *Evolutionary Computation, IEEE Transactions on*, 9(2):143–158. (Cited on page 52)

Shojaee, D., Helali, H., and Alesheikh, A. (2006). Triangulation for surface modelling. In *Ninth International Symposium on the 3D Analysis of Human Movement, France*. (Cited on page 74)

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308. (Cited on pages 22 and 23)

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088. (Cited on page 63)

Snyman, J. A. (2005). *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer. (Cited on page 24)

Soncini-Sessa, R., Castelletti, A., and Weber, E. (2003). A dss for planning and managing water reservoir systems. *Environmental Modelling and Software*, 18(5):395–404. (Cited on pages 66 and 67)

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248. (Cited on pages 45 and 49)

Stulp, F. and Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*. (Cited on page 24)

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press. (Cited on pages ii, 6, 11, 14, and 77)

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063. (Cited on page 22)

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers. (Cited on pages ii, 6, 11, 14, and 20)

Tamiz, M., Jones, D., and Romero, C. (1998). Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of operational research*, 111(3):569–581. (Cited on page 43)

Tesauro, G., Das, R., Chan, H., Kephart, J., Levine, D., Rawson, F., and Lefurgy, C. (2007). Managing power consumption and performance of computing systems using reinforcement learning. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *NIPS'07*, pages 1–8. (Cited on pages ii, vi, 7, 59, 61, 66, 67, 92, and 99)

Thiéry, C. (2010). *Itération sur les politiques optimiste et apprentissage du jeu de Tetris*. PhD thesis, Nancy 1. (Cited on pages xvii and 21)

Tušar, T. (2007). Design of an algorithm for multiobjective optimization with differential evolution. (Cited on pages xviii and 40)

Uhlig, S. (2005). A multiple-objectives evolutionary perspective to interdomain traffic engineering. *International Journal of Computational Intelligence and Applications*, 5(02):215–230. (Cited on page 52)

Ullman, J. D. (1975). NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393. (Cited on page 99)

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. (2010). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84:51–80. (Cited on pages vii, xx, 8, 58, 61, 63, 85, 86, 87, 93, and 94)

van Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*, pages 207–251. Springer. (Cited on page 22)

Van Moffaert, K., Drugan, M. M., and Nowé, A. (2013). Hypervolume-based multi-objective reinforcement learning. In *Evolutionary Multi-Criterion Optimization*, pages 352–366. Springer. (Cited on pages 57, 58, and 63)

Van Veldhuizen, D. (1999). Multiobjective evolutionary algorithms: classifications, analyses, and new innovations. Technical report, DTIC Document. (Cited on page 47)

Wang, W. and Sebag, M. (2012). Multi-objective Monte-Carlo Tree Search. In *Asian Conference on Machine Learning*. (Cited on pages 57 and 58)

Wang, Y., Audibert, J., and Munos, R. (2008). Algorithms for infinitely many-armed bandits. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *NIPS'08*, pages 1–8. (Cited on page 31)

Wang, Y. and Gelly, S. (2007). Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *CIG'07*, pages 175–182. Ieee. (Cited on page 101)

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292. (Cited on page 22)

Weinstein, A. and Littman, M. L. (2012). Bandit-based planning and learning in continuous-action markov decision processes. In *ICAPS*. (Cited on page 32)

Yang, Z. and Wen, K. (2010). Multi-objective optimization of freeway traffic flow via a fuzzy reinforcement learning method. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–530. IEEE. (Cited on pages 66 and 67)

Yu, J., Buyya, R., and Ramamohanarao, K. (2008). *Workflow Scheduling Algorithms for Grid Computing*, volume 146 of *Studies in Computational Intelligence*, pages 173–214. Springer. (Cited on pages ii, vii, 7, 8, 49, 52, 99, and 101)

Zadeh, L. (1963). Optimality and non-scalar-valued performance criteria. *Automatic Control, IEEE Transactions on*, 8(1):59–60. (Cited on page 41)

Zhai, G., Zhou, Y., Ye, X., and Hu, B. (2013). A method of multi-objective reliability tolerance design for electronic circuits. *Chinese Journal of Aeronautics*. (Cited on page 6)

Zhang, W. and Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer. (Cited on pages i and 3)

Zhao, S. and Jiao, L. (2006). Multi-objective evolutionary design and knowledge discovery of logic circuits based on an adaptive genetic algorithm. *Genetic Programming and Evolvable Machines*, 7(3):195–210. (Cited on page 52)

Zheng, K., Li, H., Qiu, R. C., and Gong, S. (2012). Multi-objective reinforcement learning based routing in cognitive radio networks: Walking in a random maze. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 359–363. IEEE. (Cited on pages 66 and 67)

Zhou, A., Qu, B.-Y., Li, H., Zhao, S.-Z., Suganthan, P. N., and Zhang, Q. (2011). Multi-objective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49. (Cited on page 74)

Zitzler, E. and Künzli, S. (2004). Indicator-based selection in multiobjective search. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 832–842. Springer. (Cited on page 48)

Zitzler, E., Laumanns, M., Thiele, L., Zitzler, E., Zitzler, E., Thiele, L., and Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm. (Cited on page 37)

Zitzler, E. and Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms - a comparative case study. In Eiben, A. E., Bäck, T., Schoenauer, M., and Schwefel, H., editors, *PPSN V*, pages 292–301. LNCS 1498, Springer Verlag. (Cited on pages vi, 7, 38, and 48)

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Trans. Evolutionary Computation*, 7(2):117–132. (Cited on pages vi, 7, 38, 45, and 47)

Zou, X., Chen, Y., Liu, M., and Kang, L. (2008). A new evolutionary algorithm for solving many-objective optimization problems. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(5):1402–1412. (Cited on page 53)