MORGAN & CLAYPOOL PUBLISHERS

Multi-Objective Decision Making

Diederik M. Roijers
Shimon Whiteson

SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Ronald J. Brachman and Peter Stone, *Series Editors*

Multi-Objective Decision Making

Synthesis Lectures on Artificial Intelligence and Machine Learning

General Game Playing
Michael Genesereth and Michael Thielscher
2014

Judgment Aggregation: A Primer
Davide Grossi and Gabriella Pigozzi
2014

An Introduction to Constraint-Based Temporal Reasoning
Roman Barták, Robert A. Morris, and K. Brent Venable
2014

Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms
Rina Dechter
2013

Introduction to Intelligent Systems in Traffic and Transportation
Ana L.C. Bazzan and Franziska Klügl
2013

A Concise Introduction to Models and Methods for Automated Planning
Hector Geffner and Blai Bonet
2013

Essential Principles for Autonomous Robotics
Henry Hexmoor
2013

Case-Based Reasoning: A Concise Introduction
Beatriz López
2013

Answer Set Solving in Practice
Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub
2012

Planning with Markov Decision Processes: An AI Perspective
Mausam and Andrey Kolobov
2012

Active Learning
Burr Settles
2012

Computational Aspects of Cooperative Game Theory
Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge
2011

Representations and Techniques for 3D Object Recognition and Scene Interpretation
Derek Hoiem and Silvio Savarese
2011

A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice
Francesca Rossi, Kristen Brent Venable, and Toby Walsh
2011

Human Computation
Edith Law and Luis von Ahn
2011

Trading Agents
Michael P. Wellman
2011

Visual Object Recognition
Kristen Grauman and Bastian Leibe
2011

Learning with Support Vector Machines
Colin Campbell and Yiming Ying
2011

Algorithms for Reinforcement Learning
Csaba Szepesvári
2010

Data Integration: The Relational Logic Approach
Michael Genesereth
2010

Markov Logic: An Interface Layer for Artificial Intelligence
Pedro Domingos and Daniel Lowd
2009

Introduction to Semi-Supervised Learning
XiaojinZhu and Andrew B.Goldberg
2009

Action Programming Languages
Michael Thielscher
2008

Representation Discovery using Harmonic Analysis
Sridhar Mahadevan
2008

Essentials of Game Theory: A Concise Multidisciplinary Introduction
Kevin Leyton-Brown and Yoav Shoham
2008

A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence
Nikos Vlassis
2007

Intelligent Autonomous Robotics: A Robot Soccer Case Study
Peter Stone
2007

Multi-Objective Decision Making

Diederik M. Roijers and Shimon Whiteson

www.morganclaypool.com

ISBN: 9781627059602 paperback
ISBN: 9781627056991 ebook

DOI 10.2200/S00765ED1V01Y201704AIM034

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Lecture #34
Series Editors: Ronald J. Brachman, *Yahoo! Labs*
Peter Stone, *University of Texas at Austin*
Series ISSN
Print 1939-4608 Electronic 1939-4616

Multi-Objective Decision Making

Diederik M. Roijers
University of Oxford
Vrije Universiteit Brussel

Shimon Whiteson
University of Oxford

SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING #34

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

Many real-world decision problems have multiple objectives. For example, when choosing a medical treatment plan, we want to maximize the efficacy of the treatment, but also minimize the side effects. These objectives typically conflict, e.g., we can often increase the efficacy of the treatment, but at the cost of more severe side effects. In this book, we outline how to deal with multiple objectives in decision-theoretic planning and reinforcement learning algorithms. To illustrate this, we employ the popular problem classes of *multi-objective Markov decision processes* (MOMDPs) and *multi-objective coordination graphs* (MO-CoGs).

First, we discuss different use cases for multi-objective decision making, and why they often necessitate explicitly multi-objective algorithms. We advocate a *utility-based* approach to multi-objective decision making, i.e., that what constitutes an optimal solution to a multi-objective decision problem should be derived from the available information about user utility. We show how different assumptions about user utility and what types of policies are allowed lead to different solution concepts, which we outline in a taxonomy of multi-objective decision problems.

Second, we show how to create new methods for multi-objective decision making using existing single-objective methods as a basis. Focusing on planning, we describe two ways to creating multi-objective algorithms: in the inner loop approach, the inner workings of a single-objective method are adapted to work with multi-objective solution concepts; in the outer loop approach, a wrapper is created around a single-objective method that solves the multi-objective problem as a series of single-objective problems. After discussing the creation of such methods for the planning setting, we discuss how these approaches apply to the learning setting.

Next, we discuss three promising application domains for multi-objective decision making algorithms: energy, health, and infrastructure and transportation. Finally, we conclude by outlining important open problems and promising future directions.

KEYWORDS

artificial intelligence, decision theory, decision support systems, probabilistic planning, multi-agent systems, multi-objective optimization, machine learning

Contents

Preface

Many real-world decision problems have multiple, possibly conflicting, objectives. For example, an autonomous vehicle typically wants to minimize both travel time and fuel costs, while maximizing safety; when seeking medical treatment, we want to maximize the probability of being cured, but minimize the severity of the side-effects, etcetera.

Although interest in multi-objective decision making has grown in recent years, the majority of decision-theoretic research still assumes only a single objective. In this book, we argue that multi-objective methods are underrepresented and present three scenarios to justify the need for explicitly multi-objective approaches. Key to these scenarios is that, although the utility the user derives from a policy—which is what we ultimately aim to optimize—is scalar, it is sometimes impossible, undesirable, or infeasible to formulate the problem as single-objective at the moment when the policies need to be planned or learned. We also present the case for a *utility-based* view of multi-objective decision making, i.e., that the appropriate multi-objective solution concept should be derived from what we know about the user's utility function.

This book is based on our research activities over the years. In particular, the survey we wrote together with Peter Vamplew and Richard Dazeley [Roijers et al., 2013a] forms the basis of how we organize concepts in multi-objective decision making. Furthermore, we use insights from our work on multi-objective planning over the years, particularly in the context of the PhD research of the first author [Roijers, 2016]. Another important source for writing this book were the lectures we gave on the topic at the University of Amsterdam, and the tutorials we did at the IJCAI-2015 and ICAPS-2016 conferences, as well as the EASSS-2016 summer school.

Aim and Readership This book aims to provide a structured introduction to the field of multi-objective decision making, and to make the differences with single-objective decision theory clear. We hope that, after reading this book, the reader will be equipped to conduct research in multi-objective decision-theory or apply multi-objective methods in practice.

We expect our readers to have a basic understanding of decision theory, at a graduate or undergraduate level. In order to remain accessible to a wide range of readers, we provide intuitive explanations and examples of key concepts before formalizing them. In some cases, we omit detailed proofs of theorems in order to better focus on the intuition behind and implications of these theorems. In such cases, we provide references to the detailed proofs.

Outline This book is structured as follows. In Chapter 1, we motivate multi-objective decision making by providing examples of multi-objective decision problems and scenarios that require explicitly multi-objective solution methods. In Chapter 2, we introduce two popular classes of decision problems that we use throughout the book to illustrate specific algorithms and general

theoretical results. In Chapter 3, we present a taxonomy of solution concepts for multi-objective decision problems. Using this taxonomy, we discuss different solution methods. First, we assume that the model of the environment is known to the agents, leading to a *planning* setting. In Chapters 4 and 5, we discuss two different approaches for finding a *coverage set* using planning algorithms. In Chapter 6, we remove the assumption that the agents are given a model of the environment, and consider cases where they must learn about the environment through interaction. Finally, we discuss several illustrating applications in Chapter 6, followed by conclusions and future work in Chapter 8.

Diederik M. Roijers and Shimon Whiteson
April 2017

Acknowledgments

This book is based on our research on multi-objective decision making over the years. During this research, we collaborated with people whose input has been essential to our understanding of the field. We would like to thank several of them explicitly.

Together with Peter Vamplew and Richard Dazeley we wrote our 2013 survey article on multi-objective sequential decision making. The discussions we had about the nature of multi-objective decision problems were vital in shaping our ideas about this field, and lay the foundation for how we view multi-objective decision problems.

In the past few years, one of our main collaborators (and Diederik's other PhD supervisor), has been Frans A. Oliehoek. Together, we developed many algorithms for multi-objective decision making, including the CMOVE and OLS algorithms that we discuss in Chapters 4 and 5. Frans's vast expertise on partially observable decision problems and limitless capacity for generating new ideas have been invaluable to our work in the field of multi-objective decision making.

Together with Joris Scharpff, Matthijs Spaan, and Mathijs de Weerdt, we worked on the traffic network maintenance planning problem (which we discuss in Section 7.3), and in this context improved upon the original OLS algorithm (Chapter 5). We enjoyed this productive collaboration.

We would also like to thank our other past and present co-authors and collaborators who we have worked with on multi-objective decision making problems: Alexander Ihler, João Messias, Maarten van Someren, Chiel Kooijman, Maarten Inja, Maarten de Waard, Luisa Zintgraf, Timon Kanters, Philipp Beau, Richard Pronk, Carla Groenland, Elise van der Pol, Joost van Doorn, Daan Odijk, Maarten de Rijke, Ayumi Igarashi, Hossam Mossalam, and Yannis Assael.

Finally, we would like to thank several people with whom we had interesting discussions about multi-objective decision making over the years: Ann Nowé, Kristof van Moffaert, Tim Brys, Abdel-Illah Mouaddib, Paul Weng, Grégory Bonnet, Rina Dechter, Radu Marinescu, Shlomo Zilberstein, Kyle Wray, Patrice Perny, Paolo Viappiani, Pascal Poupart, Max Welling, Karl Tuyls, Francesco Delle Fave, Joris Mooij, Reyhan Aydoğan, and many others.

Diederik M. Roijers and Shimon Whiteson
April 2017

Table of Abbreviations

Abbreviation	Full Name	Location
AOLS	approximate optimistic linear support	Algorithm 5.10, Section 5.5
CCS	convex coverage set	Definition 3.7, Section 3.2.2
CH	convex hull	Definition 3.6, Section 3.2.2
CHVI	convex hull value iteration	Section 4.3.2
CLS	Cheng's linear support	Section 5.3
CMOVE	multi-objective variable elimination	Section 4.2.3
CoG	coordination graph	Definition 2.4, Section 2.2.1
CS	coverage set	Definition 3.5, Section 3.2
f	scalarization function	Definition 1.1, Section 1.1
MDP	Markov decision process	Definition 2.6, Section 2.3.1
MO-CoG	multi-objective coordination graph	Definition 2.5, Section 2.2.2
MODP	multi-objective decision problem	Definition 2.2, Section 2.1
MOMDP	multi-objective Markov decision process	Definition 2.8, Section 2.3.2
MORL	multi-objective reinforcement learning	Chapter 6
MOVE	multi-objective variable elimination	Algorithm 4.5, Section 4.2.3
MOVI	multi-objective value iteration	Section 4.3.2
OLS	optimistic linear support	Algorithm 5.8, Section 5.3
OLS-R	optimistic linear support with reuse	Algorithm 5.11, Section 5.6
PMOVI	Pareto multi-objective value iteration	Section 4.3.2
PCS	Pareto coverage set	Definition 3.11, Section 3.2.4
PMOVE	Pareto multi-objective variable elimination	Section 4.2.3
POMDP	partially observable Markov decision process	Section 5.2.1
PF	Pareto front	Definition 3.10, Section 3.2.4
SODP	single-objective decision problem	Definition 2.1, Section 2.1
U	undominated set	Definition 3.4, Section 3.2
VE	variable elimination	Algorithm 4.4, Section 4.2.1
VELS	variable elimination linear support	Section 5.7
VI	value iteration	Section 4.3.1
$\mathbf{V}^{\pi}$	value vector of a policy π	Definition 2.2, Section 2.1
Π	a set of allowed policies	Definition 2.1, Section 2.1
$\succ_P$	Pareto dominance relation	Definition 3.3, Section 3.1.2

CHAPTER 1

Introduction

Many real-world decision problems are so complex that they cannot be solved by hand. In such cases, *autonomous agents* that reason about these problems automatically can provide the necessary support for human decision makers. An agent is "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors" [Russell et al., 1995]. An artificial agent is typically a computer program—possibly embedded in specific hardware—that *takes actions* in an environment that changes as a result of these actions. Autonomous agents can act without human control or intervention, on a user's behalf [Franklin and Graesser, 1997].

Artificial autonomous agents can assist us in many ways. For example, agents can control manufacturing machines to produce products for a company [Monostori et al., 2006, Van Moergestel, 2014], drive a car in place of a human [Guizzo, 2011], trade goods or services on markets [Ketter et al., 2013, Pardoe, 2011], and help ensure security [Tambe, 2011]. As such, autonomous agents have enormous potential to improve our productivity and quality of life.

In order to successfully complete tasks, autonomous agents require the capacity to reason about their environment and the consequences of their actions, as well as the desirability of those consequences. The field of *decision theory* uses probabilistic models of the environment, called *decision problems*, to formalize the tasks about which such agents reason. Decision problems can include the *states* the environment can be in, the possible *actions* that agents can perform in each state, and how the state is affected by these actions. Furthermore, the desirability of actions and their effects are modeled as numerical feedback signals. These feedback signals are typically referred to as *reward*, *utility*, *payoff*, or *cost* functions. Solving a decision problem consists of finding a *policy*, i.e., rules for how to behave in each state, that is optimal in some sense with respect to these feedback signals.

In most research on planning and learning in decision problems, the desirability of actions and their effects are codified in a *scalar* reward function [Busoniu et al., 2008, Oliehoek, 2010, Thiébaux et al., 2006, Wiering and Van Otterlo, 2012]. In such scenarios, agents aim to maximize the expected (cumulative) reward over time.

However, many real-world decision problems have multiple objectives. For example, for a computer network we may want to maximize performance while minimizing power consumption [Tesauro et al., 2007]. Similarly, for traffic control, we may want to maximize throughput, minimize latency, maximize fairness to drivers, and minimize noise and pollution. In response to a query, we may want a search engine to provide a balanced list of documents that maximizes

both the relevance to the query and the readability of the documents [Van Doorn et al., 2016]. In probabilistic planning, e.g., path planning for robots, we may want to maximize the probability of reaching a goal, while minimizing the expected cost of executing the plan [Bryce, 2008, Bryce et al., 2007]. Countless other real-world scenarios are naturally characterized by multiple objectives.

In all the cases mentioned above, the problem is more naturally expressed using a vector-valued reward function. When the reward function is vector-valued, the value of a policy is also vector-valued. Typically, there is no single policy that maximizes the value for all objectives simultaneously. For example, in a computer network, we can often achieve higher performance by using more power. If we do not know the exact preferences of the user with respect to these objectives, or indeed if these preferences may change over time, it can be crucial to produce a set of policies that offer different trade-offs between the objectives, rather than a single optimal policy.

The field of *multi-objective decision making* addresses how to formalize and solve decision problems with multiple objectives. This book provides an introductory overview of this field from the perspective of artificial intelligence. In addition to describing multi-objective decision problems and algorithms for solving them, we aim to make explicit the key assumptions that underly work in this area. Such assumptions are often left implicit in the multi-objective literature, which can be a source of confusion, especially for readers new to the topic. We also aim to synthesize these assumptions and offer a coherent, holistic view of the field.

We start by explicitly formulating the motivation for developing algorithms that are specific to multi-objective decision problems.

1.1 MOTIVATION

The existence of multiple objectives in a decision problem does not automatically imply that we require specialized multi-objective methods to solve it. On the contrary, if the decision problem can be *scalarized*, i.e., the vector-valued reward function can be converted to a scalar reward function, the problem may be solvable with existing single-objective methods. Such a conversion involves two steps [Roijers et al., 2013a]. The first step is to specify a *scalarization function* that expresses the utility of the user for different trade-offs between the objectives.

Definition 1.1 A *scalarization function* f is a function that maps a multi-objective value of a policy π of a decision problem, $\mathbf{V}^{\pi}$, to a scalar value $V^{\pi}_{\mathbf{w}}$:

$$V^{\pi}_{\mathbf{w}} = f(\mathbf{V}^{\pi}, \mathbf{w}),$$

where $\mathbf{w}$ is a weight vector that parameterizes f.

The second step of the conversion is to define a single-objective version of the decision problem such that the utility of each policy π equals the scalarized value of the original multi-objective decision problem $V^{\pi}_{\mathbf{w}}$.

Though it is rarely stated explicitly, all research on automated multi-objective decision making rests on the premise that there are decision problems for which performing one or both of these conversion steps is impossible, infeasible, or undesirable at the moment at which planning or learning must be performed. Here, we discuss three scenarios, illustrated in Figure 1.1, where this is the case, and specialized multi-objective methods are therefore preferable. In these scenarios, we assume a planning setting. However, in Chapter 6, we briefly address the learning setting.

Figure 1.1a depicts the *unknown weights scenario*. In this scenario, $\mathbf{w}$ is unknown at the moment when planning must occur: the *planning phase*. For example, consider a company that mines different resources from different mines spread out through a mountain range. The people who work for the company live in villages at the foot of the mountains. In Figure 1.2, we depict the problem this company faces: in the morning, one van per village needs to transport workers from that village to a nearby mine, where various resources can be mined. Different mines yield different quantities of resource per worker. The market prices per unit of resource vary through a stochastic process and every price change can affect the optimal assignment of vans. Furthermore, the expected price variation increases with time. It is therefore critical to act based on the latest possible price information in order to maximize performance.

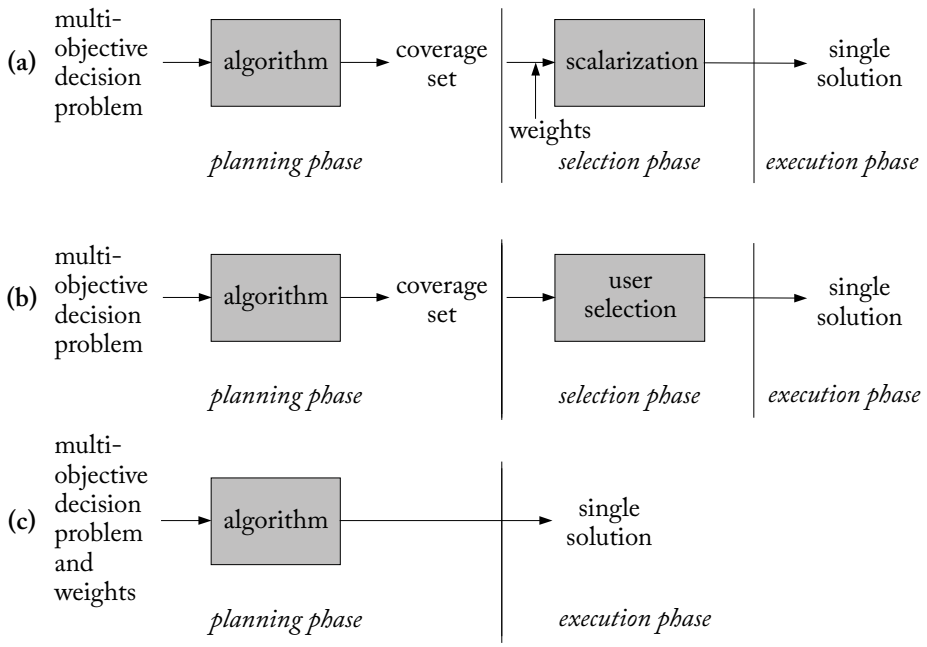

Figure 1.1: The three motivating scenarios for multi-objective decision-theoretic planning: (a) the unknown weights scenario, (b) the decision support scenario, and (c) the known weights scenario.

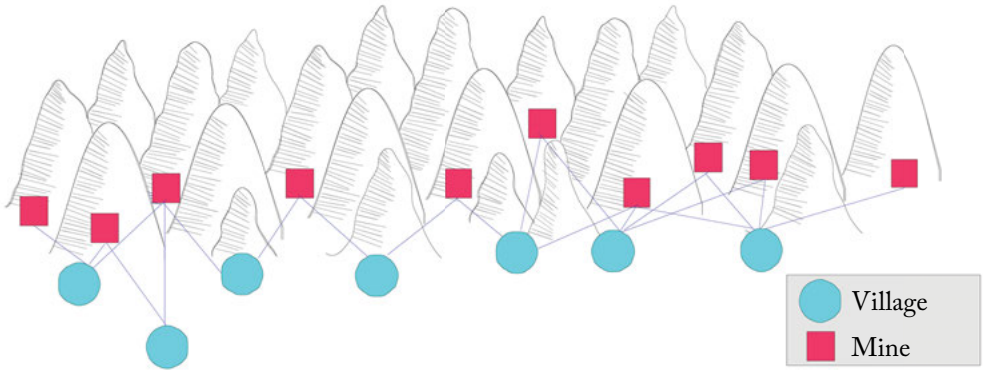

Figure 1.2: Mining company example.

Because computing the optimal van assignment takes time, computing the optimal assignment with each price change is impossible. Therefore, we need a multi-objective method that computes a set containing an optimal solution for every possible value of the prices, $\mathbf{w}$. We call such a set a *coverage set*, as it "covers" all possible preferences of the user (i.e., the possible prices) with respect to the objectives (as specified by f).[1] Although computing a coverage set is computationally more expensive than computing a single optimal policy for a given price, it needs to be done only once. Furthermore, the *planning phase* can take place in advance, when more computational resources are available.

In the *selection phase*, when the prices ($\mathbf{w}$) are revealed and we want to use as little computation time as possible, we can use the coverage set to determine the best policy by simple maximization. Finally, in the *execution phase*, the selected policy is employed.

In this book, we focus on algorithms for the planning/learning phase. For the selection phase, specialized *preference elicitation* algorithms for selecting the policy with the optimal utility from the coverage set may be necessary when the coverage set is large. For example, Chu and Ghahramani [2005], propose an approach to preference elicitation via Gaussian processes.

In the unknown weights scenario, *a priori* scalarization is impossible, because it would shift the burden of computation toward a point in time where it is not available. The scalarization f is known, and the weights $\mathbf{w}$ will become available in the selection phase, where a single policy is selected for execution.

By contrast, in the *decision support scenario* (Figure 1.1b), $\mathbf{w}$ and f are never made explicit. In fact, scalarization is infeasible throughout the entire decision-making process because of the difficulty of specifying $\mathbf{w}$ and/or f. For example, when a community is considering the construction of a new metro line, economists may not be able to accurately compute the economic benefit of reduced commuting times. The users may also have "fuzzy" preferences that defy meaningful

[1] We provide a formal definition of the term *coverage set* in Chapter 3, Definition 3.5.

quantification. For example, if construction of the new metro line could be made more efficient by building it in such a way that it obstructs a beautiful view, then a human designer may not be able to quantify the loss of beauty. The difficulty of specifying the exact scalarization is especially apparent when the designer is not a single person but a committee or legislative body whose members have different preferences and agendas, such as the politicians and interest groups involved in constructing the metro line. In such a system, the multi-objective planning method is used to calculate a coverage set with respect to the constraints that can safely be imposed on f and $\mathbf{w}$. For example, we can safely assume that gaining value in one objective, without reducing the value in any of the others cannot reduce the utility to the user (i.e., the scalarized value).

As shown in Figure 1.1b, the decision support scenario proceeds similarly to the unknown weights scenario in the planning phase. In the selection phase, however, the user or users directly select a policy from the coverage set according to their idiosyncratic preferences, rather than explicitly computing a numerical utility by applying the scalarization function to each value vector.

In the decision support scenario, one could still argue that scalarization before planning (or learning) is possible in principle. For example, the loss of beauty can be quantified by measuring the resulting drop in housing prices in neighborhoods that previously enjoyed an unobstructed view. However, the difficulty with explicit scalarization is not only that doing so may be impractical but, more importantly, that it forces the users to express their preferences in a way that may be inconvenient and unnatural. This is because selecting $\mathbf{w}$ requires weighing hypothetical trade-offs, which can be much harder than choosing from a set of actual alternatives. This is a well understood phenomenon in the field of *decision analysis* [Clemen, 1997], where the standard workflow involves presenting alternatives *before* soliciting preferences. In the same way, algorithms for multi-objective decision problems can provide critical decision support; rather than forcing the users to specify f and $\mathbf{w}$ in advance, these algorithms just prune policies that would not be optimal for any f and $\mathbf{w}$ that fit the known constraints on the preferences of the users, and produce a coverage set. Because this coverage set contains optimal solutions for all f and $\mathbf{w}$ that fit the known constraints—rather than just all $\mathbf{w}$ for a specified f, as in the unknown weights scenario—it offers a range of alternatives from which the users can select, according to preferences whose relative importance is not easily quantified.

Finally, we present one more scenario, which we call the *known weights scenario*, that requires explicit multi-objective methods. In this scenario, we assume that $\mathbf{w}$ is known at the time of planning and thus scalarization is possible. However, it may be *undesirable* because of the difficulty of the second step in the conversion. In particular, if f is nonlinear, then the resulting single-objective problem may be much more complex than the original multi-objective problem. As a result, finding the optimal policy may be intractable while the original multi-objective problem is tractable. For example, in multi-objective Markov decision processes (MOMDPs[2]), which

[2]This abbreviation is also used for *mixed-observability MDPs* [Ong et al., 2010], which we do not consider here; we use the abbreviation MOMDPs solely for multi-objective MDPs.

we formally define in Chapter 2, nonlinear scalarization destroys the *additivity property* on which single-objective solution methods rely (see Section 3.2.3).

Figure 1.1c depicts the known weights scenario. In contrast to the other scenarios, in the known weights scenario, the multi-objective method only produces one policy, which is then executed, i.e., there is no separate selection phase.

The scenarios we have presented here require explicit multi-objective methods because *a priori* scalarization of the multi-objective decision problems, and subsequent solving with standard single-objective methods, does not apply. In this book, we focus on the two multi-policy scenarios, i.e., the unknown weights and decision support scenarios, in which the goal of a multi-objective planning method is to produce a coverage set. From this coverage set, the policy that maximizes *user utility* is selected in the selection phase.

Of course, computing a coverage is in general more difficult than finding a single policy, and thus multi-objective methods are typically more expensive than their single-objective counterparts. It is natural then to wonder whether the complications of a multi-objective approach are merited. After all, many single-objective problems are already intractable. In this book, we take a pragmatic perspective: multi-objective methods are not a panacea and are not always the best option, even if the problem is naturally modeled with multiple objectives. If the scalarization weights are known (or can be reasonably estimated) before planning begins, and *a priori* scalarization does not yield an intractable problem, then converting a multi-objective problem to a single-objective one may be the best option. However, in any of the many cases where such a conversion is not possible or practical, then the multi-objective methods discussed in this book may prove indispensable.

1.2 UTILITY-BASED APPROACH

The goal of solving all—including multi-objective—decision problems is to maximize user utility. However, in the unknown weights and decision support scenarios, we cannot optimize user utility directly because, at the time when planning or learning takes place, the parameters $\mathbf{w}$ of the scalarization function f, which maps the multi-objective values to a scalar utility, are unknown. Therefore, we must compute a *coverage set*: a set of policies such that, for every possible scalarization, a maximizing policy is in the set (see Definition 3.5).

In this book, we argue that which policies should be included in the coverage set should be derived from what we know about f. We call this the *utility-based approach*, in contrast to the *axiomatic approach* that axiomatically assumes the coverage set is the *Pareto front*, which we define formally in Chapter 3. In short, the Pareto front is the set of all policies for which there is no other policy that has at least equal value in all objectives and has a higher value in at least one objective. Indeed, the Pareto front contains at least one optimal policy for most, if not all, scalarization functions that occur in practice. However, we argue that, while the Pareto front is *sufficient*, it is often not *necessary*. In fact, we show in Chapter 3 that the full Pareto front is required only in the special case where the scalarization function is nonlinear and policies must be deterministic.

A utility-based approach thus often results in a much smaller coverage set, which is typically less expensive to compute and easier for the user to examine.

Another advantage of the utility-based approach is that it is possible to derive how much utility is maximally lost when an exact coverage set cannot be computed [Zintgraf et al., 2015]. Such bounds are often crucial for a meaningful interpretation of the quality of approximate methods for decision-theoretic planning or learning, especially when comparing algorithms [Oliehoek et al., 2015]. Furthermore, the bounds provide insight into the quality and reliability of the selected final policy.

CHAPTER 2

Multi-Objective Decision Problems

In this chapter, we introduce the concept of a multi-objective decision problem. Then we describe two concrete classes of multi-objective decision problems that we use throughout the book: multi-objective coordination graphs and multi-objective Markov decision processes. However, before introducing the concrete multi-objective decision problems, we first introduce their single-objective counterparts.

2.1 MULTIPLE OBJECTIVES

In this book, we focus on different (cooperative) multi-objective decision problems. Multi-objective decision problems contrast single-objective decision problems, which are more common in the literature. In their most general form, single-objective decision problems can be defined as a set of policies and a value function that we wish to maximize:

Definition 2.1 A cooperative *single-objective decision problem (SODP)*, consists of:

- a set of allowed (joint) *policies* Π,
- a value function that assigns a real numbered value, $V^{\pi} \in \mathbb{R}$, to each joint policy $\pi \in \Pi$, corresponding to the desirability, i.e., the utility, of the policy.

Definition 2.2 In a cooperative *multi-objective decision problem (MODP)*, Π is the same as in an SODP, but

- there are $d \geq 1$ *objectives*, and
- the value function assigns a *value vector*, $\mathbf{V}^{\pi} \in \mathbb{R}^d$, to each joint policy $\pi \in \Pi$, corresponding to the desirability of the policy *with respect to each objective*.

We denote the value of policy π in the i-th objective as V_i^{π}.

Both $\mathbf{V}^{\pi}$ and Π may have particular forms that affect the way they should be computed, as we discuss in Chapter 3. For example, there may be multiple agents, each with its own action space. In such settings, a solution consists of a *joint policy* containing a *local policy* for each agent.

We assume that Π is known and that it is in principle possible to compute the value of each (joint) policy. Furthermore, we assume that, if there are multiple agents, these agents are *cooperative*.

Definition 2.3 A multi-agent MODP is *cooperative* if and only if all agents get the same (team) value, $\mathbf{V}^{\pi}$, for executing a joint policy $\pi \in \Pi$, i.e., there are no individual rewards. A single-agent MODP is cooperative by default.

This definition of cooperative is common in the field of decision theory, e.g., in multi-agent MDPs [Boutilier, 1996, Scharpff et al., 2016] and Dec-POMDPs [Oliehoek and Amato, 2016]. However, the term "cooperative" is used differently in cooperative game theory [Chalkiadakis et al., 2011, Igarashi and Roijers, 2017], in which agents form coalitions on the basis of their individual utilities. In this book, we consider only decision problems that are cooperative according to Definition 2.3.

In an SODP, the value function provides a complete ordering on the joint policies, i.e., for each pair of policies π and π', V^{π} must be greater than, equal to, or less than $V^{\pi'}$. By contrast, in an MODP, the presence of multiple objectives means that the value function $\mathbf{V}^{\pi}$ is a vector rather than a scalar. Such value functions supply only a partial ordering. For example, it is possible that, $V_i^{\pi} > V_i^{\pi'}$ but $V_j^{\pi} < V_j^{\pi'}$. Consequently, unlike in an SODP, we can no longer determine which values are optimal without additional information about how to prioritize the objectives, i.e., about what the *utility* of the user is for different trade-offs between the objectives.

In the *unknown weights* and *decision support scenarios* (Figure 1.1), the parameters of the scalarization function $\mathbf{w}$, or even f itself, are unknown during the planning or learning phases. Therefore, an algorithm for solving an MODP should return a set of policies that contains an optimal policy for each possible $\mathbf{w}$. Given such a solution set, the user can pick the policy that maximizes her utility in the *selection phase*. We want the solution set to contain at least one optimal policy for every possible scalarization (in order to guarantee optimality), but we also want the solution set to be as small as possible, in order to make the selection phase as efficient as possible. We discuss which solution sets are optimal, and how this can be derived from different assumptions about the scalarization function f (Definition 1.1), and the set of permitted policies Π in the MODP in Chapter 3. In the rest of this section, we introduce two different MODP problem classes.

2.2 MULTI-OBJECTIVE COORDINATION

The first class of MODPs that we treat is the *multi-objective coordination graph (MO-CoG)*.[1] In a MO-CoG, multiple agents need to coordinate their actions in order to be effective. For example, in the mining problem of Figure 1.2, each agent represents a van with workers from a single village. Each of these vans can go to different mines within reasonable traveling distance, leading

[1] In the literature, MO-CoGs have many different names: *multi-objective weighted constraint satisfaction problems (MO-WCSPs)* [Rollón, 2008], *multi-objective constraint optimization problems (MO-COPs)* [Marinescu, 2011], and *multi-objective collaborative graphical games (MO-CoGG)* [Roijers et al., 2013c].

to a set of different possible actions for each agent. Each mine yields a different expected amount of gold (the first objective) and silver (the second objective). Because mining can be done more efficiently when more workers are present at a mine, it is vitally important that the different agents (i.e., vans) coordinate which mines they go to.

Other examples of problems that can be modeled as a MO-CoG are: risk-sensitive combinatorial auctions, in which we want to maximize the total revenue, while minimizing the risk for the auctioneer [Marinescu, 2011], and maintenance scheduling for offices in which the energy consumption, costs, and overtime for the maintenance staff must all be minimized [Marinescu, 2011].

2.2.1 SINGLE-OBJECTIVE COORDINATION GRAPHS

Before we formally define MO-CoGs, we first define the corresponding single-objective problem, i.e., *coordination graphs* (CoGs) [Guestrin et al., 2002, Kok and Vlassis, 2004]. In the context of coordination graphs, the notion of reward is typically referred to as *payoff* in the literature. Payoff is usually denoted u (for *utility*). We adopt this terminology and notation.

Definition 2.4 A *coordination graph* (CoG) [Guestrin et al., 2002, Kok and Vlassis, 2004] is a tuple $\langle \mathcal{D}, \mathcal{A}, \mathcal{U} \rangle$, where

- $\mathcal{D} = \{1, \ldots, n\}$ is the set of n agents,
- $\mathcal{A} = \mathcal{A}_i \times \ldots \times \mathcal{A}_n$ is the joint action space: the Cartesian product of the finite action spaces of all agents. A joint action is thus a tuple containing an action for each agent $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$, and
- $\mathcal{U} = \{u^1, \ldots, u^\rho\}$ is the set of ρ scalar *local payoff functions*, each of which has limited *scope*, i.e., it depends on only a subset of the agents. The total team payoff is the sum of the local payoffs: $u(\mathbf{a}) = \sum_{e=1}^{\rho} u^e(\mathbf{a}_e)$.

In order to ensure that the coordination graph is *fully cooperative*, all agents share the payoff function $u(\mathbf{a})$. We abuse the notation e both to index a local payoff function u^e and to denote the subset of agents in its scope; $\mathbf{a}_e$ is thus a *local joint action*, i.e., a joint action of this subset of agents. The decomposition of $u(\mathbf{a})$ into local payoff functions can be represented as a *factor graph* [Bishop, 2006] (Figure 2.1); a bipartite graph containing two types of vertices: agents (variables) and local payoff functions (factors), with edges connecting local payoff functions to the agents in their scope.

The main challenge in a CoG is that the size of the joint action space $\mathcal{A}$ grows exponentially with the number of agents. It thus quickly becomes intractable to enumerate all joint actions and their associated payoffs. Key to solving CoGs is therefore to exploit *loose couplings* between agents, i.e., each agent's behavior directly affects only a subset of the other agents.

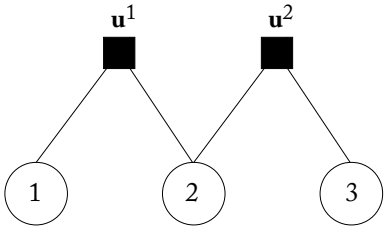

Figure 2.1: A CoG with three agents and two local payoff functions. The factor graph illustrates the loose couplings that result from the decomposition into local payoff functions. In particular, each agent's choice of action directly depends only on those of its immediate neighbors, e.g., once agent 1 knows agent 2's action, it can choose its own action without considering agent 3.

Figure 2.1 shows the factor graph of an example CoG in which the team payoff function decomposes into two local payoff functions, each with two agents in scope:

$$u(\mathbf{a}) = \sum_{e=1}^{\rho} u^e(\mathbf{a}_e) = u^1(a_1, a_2) + u^2(a_2, a_3).$$

The local payoff functions are defined in Table 2.1. We use this CoG as a running example throughout this book. The local payoff functions, with their limited scopes, encode the loose couplings: each agent can only directly affect another agent when they share, i.e., are both in the scope of, a local payoff function. For example, if we fix the action for agent 2 to be $\dot{a}_2$, then agents 1 and 3 can decide upon their optimal actions independently, as they do not directly affect each other.

Table 2.1: The payoff matrices for $u^1(a_1, a_2)$ (left) and $u^2(a_2, a_3)$ (right). There are two possible actions per agent, denoted by a dot ($\dot{a}_1$) and a bar ($\bar{a}_1$).

	$\dot{a}_2$	$\bar{a}_2$
$\dot{a}_1$	3.25	0
$\bar{a}_1$	1.25	3.75

	$\dot{a}_3$	$\bar{a}_3$
$\dot{a}_2$	2.5	1.5
$\bar{a}_2$	0	1

2.2.2 MULTI-OBJECTIVE COORDINATION GRAPHS

We now consider the multi-objective setting:

Definition 2.5 A *multi-objective coordination graph (MO-CoG)* [Roijers et al., 2015b] is a tuple $\langle \mathcal{D}, \mathcal{A}, \mathcal{U} \rangle$ where:

- $\mathcal{D}$ and $\mathcal{A}$ are the same as in a CoG, but,
- $\mathcal{U} = \{\mathbf{u}^1, ..., \mathbf{u}^\rho\}$ is now the set of ρ, *d-dimensional* local payoff functions. The total team payoff is the sum of local *vector-valued* payoffs: $\mathbf{u}(\mathbf{a}) = \sum_{e=1}^{\rho} \mathbf{u}^e(\mathbf{a}_e)$.

For example, payoffs for a MO-CoG structured as in Figure 2.1, i.e.,

$$\mathbf{u}(\mathbf{a}) = \mathbf{u}^1(a_1, a_2) + \mathbf{u}^2(a_2, a_3),$$

are provided in Table 2.2.

Table 2.2: The two-dimensional payoff matrices for $\mathbf{u}^1(a_1, a_2)$ (left) and $\mathbf{u}^2(a_2, a_3)$ (right)

	$\dot{a}_2$	$\bar{a}_2$
$\dot{a}_1$	(4, 1)	(0,0)
$\bar{a}_1$	(1, 2)	(3, 6)

	$\dot{a}_3$	$\bar{a}_3$
$\dot{a}_2$	(3, 1)	(1, 3)
$\bar{a}_2$	(0, 0)	(1, 1)

The only difference between a MO-CoG and a CoG is the dimensionality of the payoff functions. A CoG is thus a special case of a MO-CoG, i.e., a MO-CoG in which $d = 1$. Furthermore, when preferences are known, it may be possible to scalarize a MO-CoG and thus transform it into a CoG. For example, if we know the scalarization function is linear, i.e., $f(\mathbf{u}, \mathbf{w}) = \mathbf{w} \cdot \mathbf{u}$, and its parameter vector $\mathbf{w} = (0.75, 0.25)$ is, then we can scalarize the multi-objective payoff functions of Table 2.2 to the single-objective payoff functions of Table 2.1 before planning.

In a *deterministic policy* for a MO-CoG, the agents pick one joint action. In a *stochastic policy*, the agents draw a joint action from a probability distribution. Note that such a probability distribution is in general defined over *joint* actions, and there can thus still be coordination between the agents when the policy is stochastic.

When f and/or $\mathbf{w}$ are unknown in the planning or learning phase—as is the case in the *unknown weights* and *decision support* scenarios discussed in Section 1.1—we need to produce a set of policies that contains at least one optimal policy for each possible f and $\mathbf{w}$. The solution of a MO-CoG is thus a coverage set of policies. In general, this can contain both deterministic and stochastic policies. We explain why this is important for MO-CoGs (but not for CoGs) in Chapter 3.

2.3 MULTI-OBJECTIVE MARKOV DECISION PROCESSES

The second class of MODPs that we treat is the *multi-objective Markov decision process (MOMDP)*, in which a single agent needs to perform a sequence of actions over time. This sequence of actions typically takes place in an environment that is affected by these actions. Therefore, the agent has to consider not only its immediate reward, but also the reward it will be able obtain later in the states it visits in the future.

Consider the robot (shown as a Pacman symbol) in Figure 2.2 that needs to navigate in a socially appropriate way to reach the blue target zone in the upper right corner. We want the robot to reach the target as soon as possible, i.e., minimize the time to reach the target, but also minimize the hindrance that the robot causes to the other person by avoiding her personal space (indicated in red) along the way. By driving through the personal space of the person, it can obtain a higher value with respect to the first objective but a lower value in the second objective. Which policy is optimal thus depends on how much we value the first objective relative to the second objective.

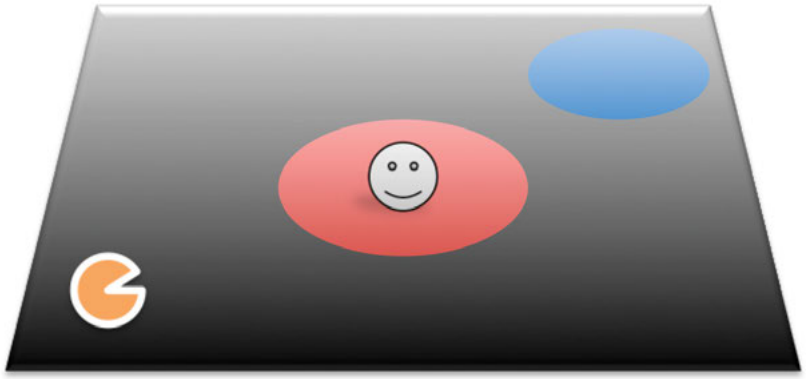

Figure 2.2: A social robot navigation problem MDP: a robot (indicated by the pacman symbol) needs to reach the target zone in the upper right corner (objective 1). However, the robots needs to avoid the personal space of the person standing in between the start position of the robot and the target zone (objective 2).

2.3.1 SINGLE-OBJECTIVE MARKOV DECISION PROCESSES

The single-objective version of an MOMDP is an MDP:

Definition 2.6 A (single-objective) *Markov decision process (MDP)* [Bellman, 1957b] is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \mu_0 \rangle$ where,

- $\mathcal{S}$ is the state space, i.e., the set of possible states the environment can be in,
- $\mathcal{A}$ is the action space, i.e., the set of actions the agent can take,
- $T : S \times A \times S \to [0, 1]$ is the transition function, giving the probability of a next state given an action and a current state,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, specifying the immediate expected scalar reward corresponding to a transition.

- μ_0 is the distribution over initial states s_0.

At each timestep t, the agent observes the current state of the environment $s \in \mathcal{S}$. When the agent takes an action $a \in \mathcal{A}$ the environment transitions to a new state s'.

The state in an MDP is *Markovian*, i.e., the current state s of the environment and the current action of the agent a are a sufficient statistic for predicting the next transition probabilities $T(s'|s,a)$ and the associated expected immediate reward. The agent's history, i.e., the states and actions that led to the current state, do not provide additional information in that respect.

The agent's goal in an MDP is to find a policy π that maximizes the return R_t, which is some function of the rewards received from timestep t and onward. In the broadest sense, a policy can condition on everything that is known to the agent.

A *state-indepedent value function* V^π specifies the expected return when following π from the initial state:

$$V^\pi = E[R_0 \mid \pi, \mu_0]. \tag{2.1}$$

We further specify first the return, R_t, and then the policy, π.

Typically, the return is *additive* [Boutilier et al., 1999], i.e., it is a sum of the rewards attained at each timestep. When the returns are additive, V^π becomes an expectation over this sum. In a *finite-horizon setting* there is a limited number of timesteps, h, and the sum is typically undiscounted:

$$V^\pi = E[\sum_{t=0}^{h-1} R(s_t, a_t, s_{t+1})|\pi, \mu_0]. \tag{2.2}$$

In a *discounted infinite-horizon setting*, the number of timesteps is not limited, but there is a discount factor, $0 \leq \gamma < 1$, that specifies the relative importance of future rewards with respect to immediate rewards:

$$V^\pi = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})|\pi, \mu_0]. \tag{2.3}$$

Which action a_t is chosen by the agent at each timestep t depends on its policy π. If the policy is *stationary*, i.e., it conditions only on the current state, then it can be formalized as $\pi : S \times A \to [0, 1]$: it specifies, for each state and action, the probability of taking that action in that state. We can then specify the *state value function* of a policy π:

$$V^\pi(s) = E[R_t \mid \pi, s_t = s],$$

for all t when $s_t = s$. The *Bellman equation* restates this expectation recursively for stationary policies:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^\pi(s')]. \tag{2.4}$$

Note that the Bellman equation, which forms the heart of most standard solution algorithms such as *dynamic programming* [Bellman, 1957a] and *temporal difference methods* [Sutton and Barto,