Thèse présentée
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de doctorat du département d'informatique et de génie logiciel
pour l'obtention du grade de Philosophiae Doctor (Ph. D.)

# Résumé

L'allocation de ressources est un problème omniprésent qui survient dès que des ressources limitées doivent être distribuées parmi de multiples agents autonomes (e.g., personnes, compagnies, robots). La planification stochastique est aussi un problème très commun qui focalise sur le développement de modèles et d'algorithmes pour se comporter de façon optimale dans un environnement incertain. Le but de cette thèse est de proposer des algorithmes rapides et efficaces pour allouer des ressources consommables et non consommables à des agents dont les préférences sur ces ressources sont induits par un processus stochastique. Afin d'y parvenir, nous avons développé des nouveaux modèles pour des problèmes de planifications, basés sur le cadre des Processus Décisionnels de Markov (MDPs), où l'espace d'action possible est explicitement paramétrisées par les ressources disponibles. Sachant ces modèles, nous développons des algorithmes basés sur la programmation dynamique et la recherche heuristique en temps-réel afin de générer des allocations de ressources pour des agents qui agissent dans un environnement stochastique.

En particulier, nous avons utilisé la propriété acyclique des créations de tâches pour décomposer le problème. Nous avons aussi proposé une stratégie de décomposition approximative, où les agents considèrent des interactions positives et négatives entre ainsi que les actions simultanées entre les agents. Cependant, les majeurs contributions de cette thèse est l'adoption de la recherche heuristique en temps-réel pour l'allocation de ressource. Pour cette fin, la Q-decomposition et des bornes strictes sont proposées afin de diminuer drastiquement le temps de planification pour formuler une politique optimale. Ces bornes strictes permettent aussi d'élaguer l'espace d'action pour les agents.

Nous montrons analytiquement et empiriquement que les approches proposées mènent à des (dans plusieurs cas, exponentiel) diminutions dans la complexité de calcul par rapport des approches de planification standards. Finalement, nous avons testé la recherche heuristique en temps réel dans le simulateur SADM, qui est l'état de l'art d'un simulateur d'allocation de ressource pour une frégate.

# Abstract

Resource allocation is a ubiquitous problem that arises whenever limited resources have to be distributed among multiple autonomous entities (e.g., people, companies, robots). Stochastic planning is also a very common problem that focuses on developing models and algorithms for behaving optimally in uncertain environments. The goal of this thesis is to propose computationally efficient algorithms for allocating consumable and non-consumable resources among agents whose preferences for these resources are induced by a stochastic process. Towards this end, we develop new models of planning problems, based on the framework of Markov decision processes (MDPs), where the action sets are explicitly parameterized by the available resources. Given these models, we design algorithms based on dynamic programming and real-time heuristic search to formulate allocations of resources for agents to act in stochastic environments.

In particular, we have used the acyclic property of task creation to decompose the problem. We have also proposed an approximative decomposition strategy, where the agents consider positive and negative interactions as well as simultaneous actions among the agents. However, the main contributions of this thesis is the adaptation of stochastic real-time heuristic search for a resource allocation. To this end, Q-decomposition and tight bounds are proposed to diminish drastically the planning time to formulate the optimal policy. These tight bounds also enable to prune the action space for the agents.

We show analytically and empirically that our proposed approaches lead to drastic (in many cases, exponential) improvements in computational efficiency over standard planning methods. Finally, we have tested real-time heuristic search in the SADM simulator, which is a state-of-the-art simulator for the resource allocation of a platform.

# Avant-propos

J'aimerais profiter de ces quelques lignes pour remercier les personnes qui ont collaboré à l'aboutissement de cette thèse. Dans un premier temps, j'aimerais remercier mon directeur de recherche, M. Brahim Chaib-draa, pour ses précieux conseils, sa disponibilité et ses encouragements. Il a toujours été présent pour m'aider à faire avancer mes recherches. J'aimerais également remercier Abder Rezak Benaskeur pour son implication dans ces travaux. Sans lui, mon parcourt aurait été beaucoup plus difficile.

J'aimerais remercier le CRSNG, le RDDC-Valcartier et le laboratoire DAMAS pour leur support financier tout au long de mes études graduées. Par ailleurs, il m'est important de mentionner la collaboration des membres du DAMAS sans qui mes études n'auraient pas été aussi plaisantes et d'une telle qualité : Julien Laumonier, Charles Desjardins, Camille Besse, Andriy Burkov, Abdeslam Boularias, Minh Nguyen-Duc, Olivier Gagné, Jilles Dibangoye, Sébastien Chouinard, Patrick Cinq-Mars, Pierre-Luc Grégoire, Jean-Samuel Marier, Dany Bergeron, Patrick Dallaire, Sébastien Paquet, Frédérick Asselin, Patrick Beaumont, Mathieu Bergeron, Nicolas Bernier, Étienne Bolduc, Vincent Dumouchel, Simon Hallé, Marc-André Labrie, Jean-Claude Lacombe, Philippe Lefebvre, Ève Levesque, Thierry Moyaux, Jean-François Morissette, Philippe Pasquier, Mathieu Pelletier, Stéphane Ross, Martin Soucy et Ludovic Tobin.

J'aimerais remercier et dédier cette thèse à Diane, Raymond et Pascale pour leurs extraordinaires amour, compréhension, générosité et attention qu'ils ont toujours eus à mon égard.

Pierrick Plamondon

*À Diane, Raymond et Pascale*

*Là où il y a de la passion, il y a de grandes réalisations*

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

The problem of resource allocation among multiple autonomous entities (agents) is ubiquitous in the modern world. Thus, for instance, an airmail company has to allocate the right types and amount of planes to distribute the mails. A company has to distribute a limited budget among the departments of the company. A government has to distribute a limited budget among its ministers. Making the right allocation decisions in these and other similar scenarios can be of critical importance.

However, what does it means to allocate the resources in the "right" way? A good government wants to maximize the long-term welfare of the residents. An airmail wants to minimize the delivery time, the delays and maximize its profit. A company aims to satisfy the needs of its client and ultimately maximize its profit.

In all the above examples, the goal of the resource-allocation process is to maximize a measure of global utility that can be obtained by the agents in the system by using these resources. This, in turn, raises the question of what determines the value of a particular set of resources to an agent. Resources are used by the agents to pursue their goals and to obtain rewards on achieving the latter.

However, in order to be able to achieve those goals, the agents often need to solve a nontrivial planning problem. For example, when an airmail allocates the right types and amount of planes to distribute the mail, the set of possible plane types and amount should be determined a priori. Similarly, the government has to think about the possible budget types that may be allocated to its ministers. In any realistic domain, such a planning process is complicated by the fact that an agent faces multiple interdependent objectives, whose achievement requires executing sequences of actions whose outcomes are uncertain. For instance, profitability of an airmail business is subject to many external factors (plane durability, demand, competition) that can seldom be predicted with certainty.

The focus of this dissertation is on the interconnected problems of resource allocation and decision making under uncertainty about the dynamics of the environments the

agents operate in. When viewed primarily from the resource allocation perspective, this work can be characterized as a study of algorithms for resource allocation where the values of the resources allocated to the agents are defined by the agents' stochastic planning problems. Alternatively, if the planning problem is placed at the forefront, the work can be described as a study of multiagent planning under uncertainty, where the interactions between agents are defined by the resources that are being distributed among them.

In this thesis, we focus on problems where the allocation of resources is done in multiple steps: the agents execute an action, then observe the result, and they execute another action, and so on, until a final state is reached. This dissertation (a) describes a formal framework of such resource-allocation and stochastic planning-problems, (b) analyzes their properties, and (c) develops tractable algorithms for computing them.

## 1.1 Resource Allocation and Stochastic Planning

The problem of resource allocation to tasks among multiple agents arises in countless domains and is studied in many diverse research fields such as economics, operations research, and computer science. The main focus of the work done in the area of resource allocation is on developing mechanisms that distribute the resources among the agents in desirable ways, given the agents' preferences over sets of resources. In such problems, the characteristics of the agents' utility functions often have a significant bearing on the properties of the resource-allocation problem. However, although defining classes of utility functions that lead to well-behaved resource-allocation problems is a topic that has received a lot of attention, most work stays agnostic about the underlying processes that define the agents' preferences for resources.

Stochastic planning, or sequential decision making under uncertainty, is also a very widely studied problem that has found application in many diverse areas. As a result, several formal mathematical frameworks (e.g., Markov decision processes (MDPs)) have emerged as popular tools for studying such problems. However, for the most part, such models do not have an explicit notion of resources and do not explicitly address the problem of planning under resource constraints.

The fundamental insight of the work in this dissertation is that these two classes of problems are strongly intertwined in ways that make analyzing and solving them in concert very beneficial. The motivation behind this work is that many real-world domains have both resource-allocation and stochastic-planning components to them, and the main hypothesis of this thesis is that by integrating these two problems and studying them in tandem, we can fruitfully exploit structure that is lost if the problems are considered in isolation. For example, real-time heuristic search is a planning

component, and when tight bounds for resource allocation are defined, this integration permits a significant reduction in planning time. Indeed, this dissertation supports its claims with analytical and empirical data.

## 1.2 Main Contributions

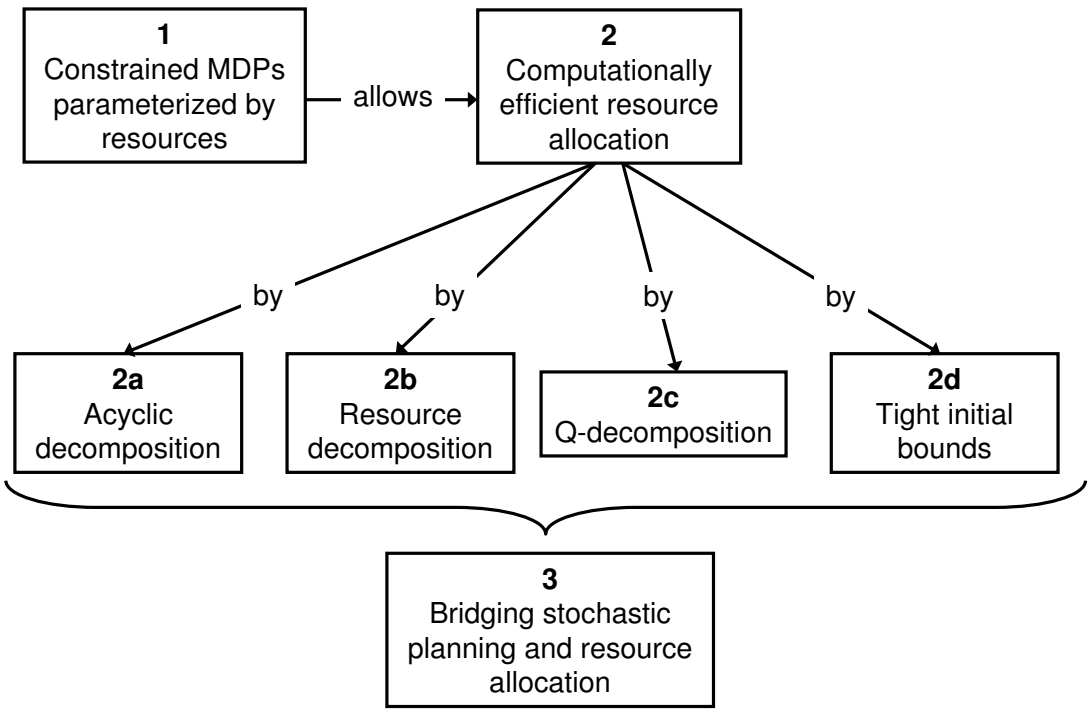



Figure 1.1: Main Contributions of this thesis.

The main contribution of this dissertation is a class of new resource-allocation methods for problems where agents' utility functions are induced by Markov decision processes. The main result of this work is based on the fact that there is a lot of structure in such MDP induced preferences, which can be exploited to yield drastic (often exponential) reductions in computational complexity of the resource-allocation algorithm.

More specifically, the major contributions of the work presented in this dissertation are as follows (depicted schematically in Figure 1.1 with the labels in the figure corresponding to the numbering in the list below).

1. **Markov decision processes with resources constraints.** In this work, we present new models based on the framework of Markov decision processes (MDPs) of stochastic-planning problems for agents whose capabilities are parameterized by the resources available to them. These models may capture situations where the

agents have limited capacities that restrict what sets of resources they can make use of. The benefit of making the notion of resources and capacities explicit in the stochastic planning models is that it allows a parameterization of the planning problem that supports the development of efficient multiagent resource-allocation methods.

2. **Computationally efficient resource allocation.** We develop and evaluate a suite of computationally efficient resource-allocation methods for agents with preferences induced by MDPs. The computational efficiency is achieved through the use of the following techniques.

   (a) **Acyclic decomposition.** The idea of acyclic decomposition is to transform our resource allocation problem in an abstract acyclic one, which contains many cyclic components. A component corresponds to a group of task, and the graph contains a component for each possible task combination. The formation of the acyclic graph is induced by task creations by other tasks. This graph is solved from the leaf to the root and permits a significant reduction in planning time.

   (b) **Resource decomposition.** The resource allocation problem is decomposed so that a planning agent manages each specific resource. The separate policies produced by the agents, if not coordinated, are not optimal for two reasons. Firstly, some resources may have positive and negative interactions since the expectation of realizing a certain task $ta_1$ by resource $res_1$ is changed when allocating another resource $res_2$ simultaneously on task $ta_1$. Secondly, the resources have to be distributed efficiently among the tasks to accomplish. Thus, the planning agents are coordinated together during the planning process through a *central* agent, for producing a near-optimal policy. The planning agents generate a policy to allocate their resources using a MDP. This method is called "Multiagent Task Associated Markov Decision Process".

   (c) **Q-decomposition.** In some cases, the resources are already shared among the agents, but the actions made by an agent may influence the reward obtained by at least another agent. To solve efficiently this kind of problem, we use Q-decomposition in the context of real-time heuristic search. Q-decomposition permits to reduce significantly the theoretical and practical complexity of the problem to formulate an optimal policy for the agents.

   (d) **Tight initial bounds.** We propose tight initial lower and upper bounds for real-time dynamic programming in the context of a resource allocation problem with consumable and non-consumable resources. These bounds permit to diminish the number of backups before convergence and to prune the ac-

tion space of uncompetitive actions. Thus, these tights initial bounds reduce also the planning time to formulate an optimal policy.

3. **Bridging stochastic planning and resource allocation.** Another (perhaps more speculative) contribution of this thesis is that, by considering resource-allocation and planning problems concurrently, it strengthens the currently underdeveloped link between these two research areas. This dissertation only begins to explore this broader connection, but our results indicate that the relationship can be very synergistic.

## 1.3 Overview of the Thesis

The rest of this thesis is organized as follow:

- **Chapter 2** overviews scheduling which is the main modeling tool for resource allocation problems.
- **Chapter 3** begins by describing the major planning approaches in operation research and artificial intelligence. In particular, we describe a sequential decision making tool, Markov Decision Processes (MDPs), employed to model stochastic resource allocation problems.
- **Chapter 4** introduces the resource allocation problem which lead to this dissertation and the context surrounding the allocation of resources for a ship to counter anti-ship missiles are described. Afterwards, a toy problem which is very similar to our naval problem is presented and we discuss the best modeling tool for our problem. We argue that Markov Decision Processes are very suitable. Finally, our toy problem helps to understand the ensuing formulation of MDP in the context of resource allocation where consumable and non-consumable are available to execute a set of tasks in a sequential manner.
- **Chapter 5** presents stochastic real-time heuristic search. By using, real-time heuristic search, a policy is available at any time, but more importantly, a good initial heuristic may reduce the planning time significantly compared to standard planning approaches such as value iteration and policy iteration. This chapter also extends previous works on real-time heuristic search and proposes tight initial lower and upper bounds for real-time dynamic programming in the context of a resource allocation problem with consumable and non-consumable resources (Contribution **2d**).
- **Chapter 6** proposes different problem decomposition approaches to diminish the complexity of our problem. To this end:

  - **Section 6.1** deals for a specific type of resource allocation problem where the resources are already shared among the agents, but the actions made by an agent may influence the reward obtained by at least another agent. In this context, Q-decomposition in the context of real-time heuristic search is employed (Contribution **2c**).
  - **Section 6.2** discusses the idea of acyclic decomposition as described in Contribution **2a**.
  - **Section 6.3** introduces Multiagent Task Associated Markov Decision Processes (MTAMDP), which is Contribution **2b**. This section also proposes a merging of MTAMDP with acyclic decomposition to further reduce the planning time.

- **Chapter 7** describes the work done with the Surface Air Defence Model (SADM) naval simulator. In particular, an heuristic search approach has been implemented which solves efficiently the problem. However, the heuristic search approach is hindered by an inadequate models which opens future work.

- **Chapter 8** concludes with a summary of the main contributions of the thesis, its limitations, and a discussion of the questions that remain open.

The next chapter overviews different techniques for scheduling found in the literature.

# Chapter 2

# Techniques for Scheduling with Uncertainty

This chapter overviews the scheduling techniques. According to Smith et al. (2000), scheduling research has focused on large problems where there is little action choice, but the resulting ordering problem is hard. In brief, scheduling aims to allocate resources to execute tasks. In this thesis, we are interested in a specific aspect of scheduling, where there are uncertainty in the effectiveness of the resources.

## 2.1 Common Models of Scheduling

According to Davenport and Beck (2002), scheduling problems are composed of *tasks* (i.e. activities), *resources*, and *constraints*. A task has a start time, end time and duration, specifying the period of time over which execution of the task takes place. The two most important classes of constraints are *temporal constraints* and *resource constraints*. Temporal constraints expresses temporal relationship between tasks, such as one task must take place after another. Resource constraints express constraints on resource usage, such as a resource may process only one task at a time.

Two common models of scheduling which have been widely studied are the *Job Shop Scheduling Problem* and the *Resource Constrained Project Scheduling Problem*. We briefly review these two models in the remainder of this section. We latter use these perfect information models to provide a basis for understanding the ways in which reasoning about uncertainty has been introduced into scheduling.

An $n \times m$ job shop scheduling problem consists of $n$ jobs and $m$ resources. Each job consists of a set of $m$ completely ordered tasks, where each task has a duration for which it must execute and a resource which it must execute on. The complete ordering defines a set of precedence constraints, meaning that no task can begin execution until

the task that immediately precedes it in the complete ordering has finished execution. Each of the $m$ tasks in a single job requires exclusive use of one of the $m$ resources defined in the problem. No tasks that require the same resource can overlap in their execution and once a task is started it must be executed for its entire duration (i.e., no pre-emption allowed). The job shop scheduling decision problem is to decide if all tasks can be scheduled, given for each job a release date of 0 and a due date of the desired makespan, $D$, while respecting the resource and precedence constraints. The job shop scheduling decision problem is NP-complete (Garey and Johnson, 1979).

A resource constrained project scheduling problem consists of a set of $n$ tasks and $m$ resources. Each task has a defined duration and may be linked by precedence constraints to any of the other tasks. Each task requires some amount of one or more of the $m$ resources during its duration of execution. Each resource has a maximum capacity expressing the total amount of the resource that can be used at any time point by any set of tasks. As with the job shop scheduling problem, a solution to the problem is to determine, given a release date for all tasks of 0 and a due date of the desired makespan, if there exists a schedule that respects the resource capacity constraints and the precedence constraints. The decision variant of the resource constrained project scheduling problem is NP-complete (Herroelen and Demeulemeester, 1995).

More realistic models of scheduling problems include components of job shop scheduling decision problem and resource constrained project scheduling problem (Nuitjen and Aarts, 1997) as well as a variety of additional constraints (e.g. breaks during which a particular resource cannot be used, time dependant resource availability, set-up and tear-down tasks required before each activity, etc.).

It should be noted that it is seldom the case that a schedule is executed in isolation. Real world schedules depend on and are depended on by external agents such as costumers and suppliers in a supply chain of a manufacturing organization. A schedule therefore is not simply an internal recipe for a set of tasks but also a basis for communication and coordination with external agents. These external dependencies make the management of uncertainty even more critical as unexpected events that are not reacted to and contained may have an impact that far out-weights their original importance.

## 2.2 Uncertainty in Scheduling

An example from the airport ground service scheduling domain (Hildum, 1994) is given in this section to provide a concrete example of uncertainty that arises in the execution of schedules. The airport ground service scheduling domain is modeled as follows:

- We are given a master timetable of flights $F : \{F_\infty, ..., F_I\}$ and a collection of

resources $R : \{R_\infty, ..., R_R\}$. Each of the flights in $F$ requires the execution of some sequence of ground-servicing tasks from the task set $T : \{T_\infty, ..., T_T\}$ depending on the particular type of ground service requested. A *job*, comprised of some sequence of $T_i$'s: $\{T_1, ..., T_n\}$, is instantiated for each flight $F_i \in F$. All flights have a ready time, corresponding to flight arrival time, and a due date, corresponding to flight departure time. The goal of the problem is to allocate resources to tasks while satisfying the temporal and resources constraints.

There are many sources of uncertainty in this problem. The set of flights $F$ is not fixed. Airport timetables are subject to fluctuation as flights that might be canceled, delayed or modified in the course of execution. Processing time is inherent dynamic. The duration of a task may depend on the time of day it is executed or on which resource is used to precess the task. Weather conditions, both local and remote, also have a significant impact on flights arriving and leaving the airport. Delays in timetables as a result of problems at other airports can also add to uncertainty. Many ground servicing tasks take longer to complete under difficult weather conditions, and may require extra, unplanned for tasks to be executed in such situations.

## 2.3 Dealing with Uncertainty

In general, there are two approaches to dealing with uncertainty in a scheduling environment: *proactive* and *reactive* scheduling. Proactive scheduling constructs predictive schedules that account for statistical knowledge of uncertainty. Reactive scheduling involves revising or reoptimizing a schedule when an unexpected event occurs.

A scheduling system that is able to deal with uncertainty is very likely to employ both proactive and reactive techniques. A proactive technique will typically requires a, perhaps trivial, reactive component to deal with the occurrence of uncertain events during scheduling execution: obviously, the schedule, exactly as defined, cannot continue to be executed with a broken machine, therefore some reasoning on execution time is mandatory even if it is necessary to put in place a contingent schedule that was proactively computed. Furthermore, it is unlikely that it will be worthwhile to take into account all unexpected events proactively. Some will be to improbable or too minor and therefore if they do occur will have to be dealt with reactively. Similarly, a major constraint on reactive scheduling is the timeliness of the response. This requirement means that optimization at execution time is nor a realistic goal. Use of a proactive technique may however provide the reactive component with strict bounds ont the complexity of its computation while possibly allowing higher quality solutions to be found.

We describe in the remaining of this chapter some approaches to scheduling with uncertainty. Section 2.3.1 examines proactive techniques that account for uncertainty

by inserting some form of redundancy, typically extra time, into the schedule. This is followed, in Section 2.3.2, by techniques using a more formal probabilistic reasoning and then, in Section 2.3.3, by techniques which create multiple schedules to deal with different contingencies that may arise during schedule execution. Section 2.3.4 addresses approaches that more explicitly make use of both off-line (proactive) and on-line (reactive) scheduling while the final section. Finally, we discuss all these techniques in Section 2.3.5.

### 2.3.1 Redundancy-based Techniques

The main characteristic of the work reviewed in this section is the reservation of extra time and/or resources so that unexpected events during execution (e.g., resource failures, longer task durations) can be dealt with by using some of this "extra" time and resources. In particular, two different redundancy approaches have been proposed in the literature: (1) Fault tolerant real-time scheduling, (2) slack-based protection.

#### Fault Tolerant Real-time Scheduling

While there are a number of differences between scheduling problems and solution techniques typical of manufacturing and project scheduling and those typical of the scheduling of real-time systems, a critical component of providing real-time guarantees for schedule execution is the ability to cope with faults. Therefore, there has been significant work done on dealing with uncertainty for real-time systems.

Redundancy is the typical way in which fault tolerance is guaranteed. Two forms of redundancy are common:

1. Resource redundancy — Multiple identical sets of resources are used to execute multiple versions of each task in parallel.

2. Time redundancy — Time is reserved to re-execute tasks that fail.

#### Slack-based Protection

Leon et al. (1994) studied robust scheduling by redefining the evaluation function of a schedule to include an expression of robustness. Given such an evaluation function, optimal schedules can be found using traditional OR search techniques.

In order to define an evaluation function, a number of robustness measures have been developed and evaluated. The problem model used is as follows: let $S$ be a job shop schedule specifying the order in which tasks are executed on machines, and let

$M_o(S)$ be the deterministic makespan of $S$ assuming no disruption. Let the random variable $M(S)$ denote the actual makespan of $S$ in the presence of disruptions. No task can be processed during a disruption and disrupted tasks must be restarted from the beginning. When disruption occur, schedule breakage is fixed by applying the Right Shift rule, which pushes tasks affected by the disruption forward in time, without changing the sequence of tasks processed by any machine.

The schedule delay is defined as a random variable expressing the difference between executed and deterministic schedule makespan.

$$\delta(S) = M(S) - M_o(S) \tag{2.1}$$

Since $M_o(S)$ is deterministic, we can write the expected values of $M$ and $\delta$ as:

$$E[M(S)] = E[\delta(S)] + M_o(S) \tag{2.2}$$

A small value for expected delay implies that the schedule is affected little by random disruptions. However, zero delays can be achieved by inserting large amounts of idle time into the schedule, therefore simply minimizing expected delay is unlikely to lead to usable schedules. Expected makespan is also important in maximizing the use of the resources in the scheduling environment, therefore, the authors define schedule robustness as a linear combination of expected makespan and delay. Let $r$ be a real value weight in the interval $[0, 1]$. Schedule robustness, $R(S)$, is defined as:

$$R(S) = r \times E[M(S)] + (1 - r) \times E[\delta(S)] \tag{2.3}$$

The $R(S)$ robustness measure is one of the few attempts outside the real-time scheduling community to formalize the definition of schedule robustness. Unfortunately, the measure conflates the notion of robustness with the traditional job shop optimization criteria of makespan minimization. A formalization of schedule robustness must be independent of a specific optimization criteria if it is to be useful. It is likely to be necessary to balance robustness against other measures of schedule quality and therefore including a particular optimization criteria in the definition of robustness limits its applicability.

#### Discussion on Redundancy-based Techniques

For the type of scheduling problem we are interested in this thesis, resource and time redundancies are irrelevant. We are interested in problems where the number of available resources and the time window to execute each tasks are fixed.

### 2.3.2 Probabilistic Techniques

Redundancy-based techniques address the problem of uncertainty with the assumption that adding redundancy can be a solution. Probabilistic techniques take a different

approach. Rather than starting with the assumption that redundancy is the solution, that start from the position that a key piece of information is the probability that the schedule can be executed. A *priori*, this is a diagnostic tool rather than the solution: it does not produce robust schedules, rather it allows the uncertainty in a schedule to be measured. However, once we have the ability to measure such probabilities, we may also be able to build schedules so as to maximize them. We can use, for example, probabilistic techniques to guide the amount and location of redundancy that should be inserted into a schedule?

### Probabilistic Real-time Fault Tolerant Scheduling

The standard form of real-time system fault tolerance guarantees is to assume a fault model as part of the problem definition. A schedule is created based on the fault model and the typical fault tolerant guarantee is that all due dates will be achieved as long as the faults arrive no more quickly than assumed by the fault model.

Burns et al. (1997) introduced the notion of a *probabilistic guarantee* for hard real-time systems. This probabilistic guarantee is a guarantee of schedulability with an associated probability. In particular, this means that a guarantee of 99% for a schedule does not indicate that 99% of the jobs will meet their due dates but rather that in 99% of the executions of this schedule, all jobs will meet their due dates. This is a similar notion of probabilistic customer service levels in inventory management where the level of inventory allocated to a warehouse is such that all customers orders will be met some percentage of the time.

### $\beta$-Robust Scheduling

Rather than faults as the source of uncertainty, another model considers duration or precessing time uncertainty. That is, the time that each activity must execute is not precisely known and the goal is to produce a schedule with the maximum probability of achieving a specific level of some performance measure.

Daniels and Carrillo (1997) introduced the notion of a $\beta$-robust schedule for a single-machine scheduling model with precessing time uncertainty. In particular, the authors noted that with such a source of uncertainty, it is insufficient to simply consider the mean value of a measure of schedule quality (e.g., mean flow time). In fact, the variance provides critical information. Under uncertainty, a schedule with optimal mean performance may have an extremely high variance wile a schedule with a sub-optimal mean performance may have a much lower variance. It it is important to minimize the risk of unacceptable performance rather than achieve optimal performance, the latter schedule may be preferable.

### Multiobjective Stochastic Dominance A* Search

In recent years much work has been carried out in the AI planning community on *decision theoretic planning*. This differ from classical AI planning in the following ways (Wellman, 1993):

- the effects of actions are described by probability distributions over outcomes;
- objectives are described by utility functions;
- the criteria for effective plan generation is expected utility maximization.

Although this work seems applicable to scheduling, and number of researchers have suggested using decision theory to deal with uncertainty in the scheduling domain (Drummond et al. (1994); Pape (1991)), little work has been carried out in this area. Here, we look at work that applies multiobjective stochastic dominance A* search on a single-machine scheduling problem. In Sections 2.3.3 and 2.3.4 we look at other work within decision theoretic planning with contingent and off-line/on-line approaches to uncertainty, respectively.

Wurman and Wellman (1996) addressed studied the stochastic lot-sizing problem which consists of a set of orders for different inventories and a task schema defining production and shipping tasks for different quantities of each inventory and for setting up the machine to produce each inventory. The optimization criteria is the expected weighted number of tardy jobs. The processing time of each production task is stochastic but proportional to the amount of inventory produced. Shipping tasks are instantaneous and do not require the machine while setup tasks have a stochastic processing time independent of the preceding and succeeding production tasks. Each order has a due date and a penalty that must be paid if the inventory is not shipped by the due date.

The authors adopted a state encoding that consists of a set of orders (and whether they have already been met), the quantity of each inventory that currently exists, and the current machine setup. The cost of a state is represented by a pair of distributions: one for the accrued cost and one for the time.

The authors showed that this state encoding requires the use of Multiobjective Stochastic Dominance A* (MO-SDA*) which ensures a sound and complete search of the state space. The precise definition of MO-SDA* is beyond the scope of this thesis, however, it should be noted that the critical contribution of the method is a necessary and sufficient basis on which paths in the search can be pruned.

#### Discussion on Probabilistic Techniques

Probabilistic techniques are essential tools for the type of resource allocation problem we are interested in this thesis. In particular, we are interested in problems where the action outcomes are stochastic, thus decision-theoretic planning is essential.

### 2.3.3 Contingent Scheduling

Redundancy and probabilistic techniques are based on the approach of generating a single schedule that is likely to be able to incorporate unexpected events without major disruptions. A different approach is represented by contingent scheduling techniques. These techniques are based on attempting to anticipate likely disruptive events and generating multiple schedules (or schedule fragment) which optimally respond to the anticipated events. This is all done *a priori* so that at execution time a set of schedules are available. Responding to unexpected (but anticipated) events and execution time simply consists of switching to the schedule that corresponds to the events that have occurred.

#### Just-in-Case Scheduling

Just-in-Case (JIC) scheduling (Drummond et al., 1994) is a technique for generating schedules in a domain where activities have uncertain durations, which can lead to schedule breakage. It has been developed and applied in the domain of telescope observation scheduling.

The telescope observation problem consist of a single resource, the telescope. Each job is an observation task, which has a time window determined by the possible observation times for the task. The task durations are stochastic, modeled using a normal distribution using statistics from previous execution data.

One way of dealing with duration uncertainty is to always assume the worst case: set all task durations to their longest time possible and solve the resulting deterministic duration scheduling problem. When some task finishes early, one introduces a "wait" activity to fill up the remaining time. A weakness of this technique is that resource usage may be very low as the duration is a worst-case estimate and therefore it is likely that the resources will be idle for much of the time. An alternative is to initially schedule with mean duration, to reassign the start time of tasks when some activity takes longer than its mean duration to execute. If a breakage occur, that is, if any task can no longer execute given its original time window, then rescheduling is performed. This approach also results in idle time during rescheduling. The goal then is to avoid schedule breakage without sacrificing schedule quality.

#### Markov Decision Processes

The goal of Markov Decision Processes (MDPs), in its classical formulation, is to find a *policy*, a mapping of state into actions, that maximize the objective function. MDPs are discussed in more details in Section 4.7.2.

#### Discussion on Contingent Scheduling

Contingent scheduling coupled with probabilistic scheduling permits to model a stochastic resource allocation problem early. In particular, uncertainty about actions outcome and duration can be modeled, and a fast response is generated in the case of action failures.

### 2.3.4 On-line/Off-line Approaches

The techniques discussed to this point have focussed on the generation of predictive schedules that, in some way, take into account the uncertainty in the environment. While some of them (e.g., the redundancy-based and the contingent techniques) assume that some reasoning will have to be done at the time of schedule execution (e.g., shifting tasks to take advantage of the slack time that was reserved of choosing one of the contingencies), in general they are off-line techniques, minimizing the need for and the complexity of on-line reasoning. In this section, we turn to work that has looked more explicitly at off-line/on-line algorithms.

Russell and Norvig (2003) distinguish two types of search. Firstly, an *off-line search* algorithm computes a complete solution before setting foot in the real world, and then executes the solution without recourse to their percept. In contrast, an *on-line search* agent operates by interleaving computation and action: first it takes an action, then it observes the environment and computes the next action. On-line search is a good idea in dynamic domains when there is a penalty for computing too long. On-line search is an even better idea for stochastic domains. In general, an off-line search would have to come up with an exponentially large contingency plan that considers all possible happenings, while an on-line search needs only consider what actually does happens.

The work reviewed in this section does not necessary explicitly discuss both off-line and on-line techniques but rather is considered from the perspective that both phases are necessary even if only one is discussed. For example, the work on least and delayed commitment scheduling, described in the next section, focusses on an off-line technique without explicitly detailing an accompanying on-line technique. This is because a variety of on-line techniques are possible and that the theme of the work is, in the former case, to make decision only when the appropriate level of information is available, and,

in the latter case, to solve the off-line problem so that strong theoretical limits can be placed on the amount of work required by an on-line phase. The other two pieces of work discussed below present both off-line and on-line phases from the perspective of artificial immune systems and from the perspective of decomposed Markov Decision Processes.

### Least Commitment and Delayed Commitment Scheduling

A common approach to off-line algorithms that specifically assume, but do not necessarily define, an on-line counterpart is the least or delayed commitment scheduling (Berry, 1993). In general, the two terms refer to the creation of a predictive schedule that does not completely define all characteristics of all activities to be executed. Rather, a set of constraints are added to the scheduling problem significantly narrowing the search space that needs to be explored in an on-line phase. Least and delayed commitment are based on the idea that decisions should not be taken where information is incomplete or uncertain if it can be avoided. In highly uncertain environments, it may be better to only generate predictive schedules for a short time in the future, since they are highly likely to break soon anyway. On the other hand, in more certain environments it may be useful to vary the level of commitment across the scheduling horizon. In the near term, schedules are generated with a high level of commitment, but further in the future weaker commitments are made which can be refined when more information about the state of the world is known.

For example, a least/delayed commitment approach to job shop scheduling problem is to post sequencing constraints between the tasks on each resource, rather than assigning specific start times. A single sequencing solution represents many possible start time solutions. The actual start times of the tasks can be found with a polynomial technique in the on-line phase which may take into account preferences and up-to-date operating conditions on the shop floor. The sequence solution can absorb minor variations in schedule execution; for instance a task starting later than planned as a result of an unplanned disruption may still be able to satisfy the sequencing constraints posted in the scheduling solution.

Though we do not make a distinction between least and delayed commitment scheduling in this thesis, there is often a subtle difference between the two approaches. Least commitment scheduling typically provides a guarantee that at least one solution exists in the search space defined by the predictive schedule while delayed commitment scheduling provides no such guarantee.

### Artificial Immune Systems

A very different approach to off-line/on-line algorithms for robust scheduling is a preliminary work that examines the creation of an artificial immune system (Hart and Ross, 1999). The approach rests on the conjecture that the conditions under which a new schedule is needed in response to some unexpected event in a factory are to some extend predictable. For example, there may patterns of customer orders and resource loads (perhaps depending on seasonality), there may be particular machine that breakdown, or there may be factories from which deliveries are typically late (or early). Furthermore, these patterns of events may have corresponding actions that can be taken during rescheduling to minimize their impact. The reactions are actually the partial schedules that are put in place during rescheduling. Therefore, the authors' goal, is to use genetic algorithms to evolve a set of partial schedule (based on historical schedule of a factory) that can be used as building blocks to respond to an unexpected event. These schedule pieces encode some specific domain dependent knowledge about reasonable schedule for the factory and therefore will significantly reduce the search space required for rescheduling.

A set of schedule is therefore evolved, off-line, for each resource using a genetic algorithm. The fragments are represented by a sequence of activities that can execute on that resource plus a "wildcard" activity. The fitness criteria for the evolution involves the matching of the fragment against historical schedules. A match occurs if some subsequence of tasks in a fragment is found to exist in an historical schedule. The wildcard task matches any task allowing the possibility of evolving more complex, non-contiguous pattern. The on-line phase is then to combine the evolved sequences to form schedules that are reactions to unexpected state of the factory. Since similar states are likely to have been encountered in the past, a combination of the evolved sequences is likely to encode a good schedule.

### Markov Task Sets

Another off-line/on-line approach is the work by Meuleau et al. (1998). These authors examined the modeling of resource allocation problems with Markov Decision Processes (MDPs). In order to solve the problem beyond the tractability limit of standard MDP techniques the authors developed an approach in which the problem is first decomposed so that the allocation for each target is solved to optimally independently and then the individual solutions are greedily integrated to find a good, but not necessarily optimal, global solution. More details on this approach is in Section 3.1.1.

**Discussion on On-line/Off-line Approaches**

Delayed and least commitment scheduling together with problem decomposition techniques such as Markov task sets would appear to be critical tools in dealing with uncertainty. Indeed, the form of decomposition used in the Markov task sets work can be seen as a form of delayed commitment scheduling: the integration of the sub-problems is left until execution time when the information about the success or failure of each task is available. Unless care is taken, however, these approaches can suffer from the same drawback as the contingency-based techniques: the off-line schedule may not produce enough information, for other, dependent tasks in the problem to create local schedules. This is most apparent in the Markov task sets work as the off-line schedules for each target may have little relation to the integrated schedule. The goal in the off-line phase must be to make a sufficiently detailed schedule so as to serve as a basis for dependant entities in the problem while maintaining enough flexibility to deal with disturbances at execution time. Though works reviewed in this section has begin to address this goal, it remains a challenging problem.

### 2.3.5 General Discussion

Probabilistic reasoning appears to be an important step in terms of bringing a level of formality to reasoning about uncertainty in scheduling. Even if it is shown that exact probabilistic guarantees of schedulability are intractable in practice, the very existence of a formal theory will provide significant support on which approximation techniques can be based. Of the approaches reviewed in this chapter and of which we are aware, probabilistic reasoning appears to be the only one which may be able to be developed into a formal theory.

As shown in the fault tolerant real-time scheduling field, redundancy technique can provide tools for a significant simplification of both predictive robustness measures and the complexity of rescheduling. It may be possible to achieve a high degree of formality and still allow tractable schedule generation by combining redundancy techniques with probabilistic reasoning.

There is clearly a need for both off-line, predictive/contingent techniques and on-line, reactive techniques for dealing with uncertainty. The challenge is to have an off-line technique which provides a flexible plan while having an on-line phase that can quickly take near-optimal decisions.

This chapter presented different techniques for scheduling. The next chapter is interested in the specific aspect of scheduling where there is uncertainty in the effect of each actions undertook to execute the tasks.

# Chapter 3

# Task Planning Under Uncertainty

This thesis aims to contribute to solve complex stochastic resource allocation problems. In general, resource allocation problems are known to be NP-Complete (Zhang, 2002). In such problems, a scheduling process suggests the action (i.e. resources to allocate) to undertake to accomplish certain tasks, according to the perfectly observable state of the environment. When executing an action to realize a set of tasks, the stochastic nature of the problem induces probabilities on the next visited state. In general, the number of states is the combination of all possible specific states of each task and available resources. In this case, the number of possible actions in a state is the combination of each individual possible resource assignment to the tasks. The very high number of states and actions in this type of problem makes it very complex. The next section describes the planning approaches found in the literature to efficiently allocate resources in this case.

## 3.1 Planning Approaches

Boutilier et al. (1999a) proposed a framework to classify various problems commonly studied in the planning and decision-making literature. In each case below of this framework, the modeling assumptions which define the problem class are specified. Notice that in AI, problems involving a choice of actions are often regarded as *planning* problems (Smith et al., 2000). Unfortunately, few AI planning systems could represent constraints between actions, nor perform the desired reasoning and optimization. While *scheduling* systems would have an easier representation of the time constraints and resources, most of them could not deal with the action choices. In a sense, most problems lies between planning and scheduling.

### 3.1.1 Planning Problem in the Operation Research/Decision Sciences Tradition

#### Markov Decision Processes (MDPs)

There is an extremely large body of research studying MDPs, and the basic algorithmic techniques are presented in some detail in Section 3.1.1. The most commonly used formulation of MDPs assumes full observability and stationarity, and uses as its optimality criterion the maximization of expected total reward over a finite horizon, maximization of expected total discounted reward over an infinite horizon, or minimization of the expected cost to a goal state. MDPs were introduced by Bellman (1957) and have been studied in depth in the fields of decision analysis and Operation Research, including the seminal work of Howard (1960). MDPs are very suitable to model a stochastic environment where the outcome of an agent's action is probabilistic and the environment is modified by some unpredictable exogenous events. For example, suppose that missile launcher sometimes fails to launch a Surface Air Missile (SAM). Thus, when the own platform aims to counter a threat ($ta$) with a SAM, it may or not be able to execute this action. Part of the MDP process for this scenario is illustrated in Figure 3.1.

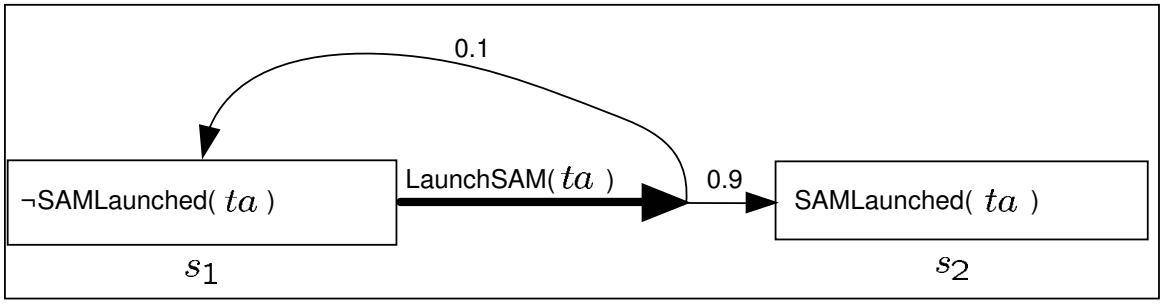



Figure 3.1: Model of an own platform which has a 90% probability to launch a SAM missile.

An MDP is thus given by:

- A state space $S$.
- Actions $A(s) \subseteq A$ applicable in each state $s \in S$.
- Transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$.
- Action costs $C(a, s) > 0$ and state rewards $R(s)$ related to the problem.

- A discount factor $\gamma$, which is a real number between 0 and 1. The discount factor describes the preference of an agent for current rewards over future rewards
- A non empty set $G \subseteq S$ of goal states.

In MDPs, the state $s'$ that results from an action $a$ is not predictable but is observable, providing feedback for the selection of the next action $a'$. As a result, a solution of an MDP is not an action sequence, but a function $\pi$ mapping states $s$ into actions $a \in A(s)$. Such a function is called a *policy*. These policies are defined to be applicable no matter what state (or distribution over states) one finds oneself in — action choices are defined for every possible state or history.

Modeling a resource allocation decision-making problem as a MDP, implies that the action space is defined by the resources available to the agent. In other words, an agent's MDP is parameterized by the available resources, and the agent's utility for a particular set of resources is defined as the expected value of the best policy that is realizable given the actions that it can execute using these resources. Furthermore, a realistic agent has inherent limitations that constrain the sets of resources it can make use of (and thus what policies it can execute). For example, an airmail is limited by the number of planes and available air routes to distribute its mail. Therefore, the problem of acting optimally under the constraints of the agent's inherent limitations arises. This problem has been studied under various contexts and using different models of agents' limitations — some that directly restrict the agents' policy or strategy spaces (Russell and Subramanian (1995); Bowling and Veloso (2004)), and some that model agents' limitations via the concept of resources (Mausam et al., 2005). The latter approach typically makes more detailed assumptions about the structure of agents' constraints, which can often be exploited in practical algorithms. As viewed from the perspective of modeling of the agents' limitations, our work falls into the latter, resource-centric category. An frequent way of dealing with the complexity of resource allocation, decomposition techniques are frequently used.

The Markov Decision Process (MDP) framework has been widely adopted by today's researchers to model a sequential and stochastic process. The choice is due to the fact that MDPs provide a well-studied and simple, yet a very expressive model of the world. Still, classical MDPs suffer from the so-called *curse of dimensionality*: the number of states grows exponentially with the number of variables that characterize the planning domain. Consequently, a polynomial time algorithm can be prohibitive for a real-time application. This chapter exposes the Markov Decision Processes (MDPs) approach, as well as modifications adopted to reduce its planning complexity.

The next sections describe in more details the MDPs characteristics. In particular, the states and state transitions, the actions, the exogenous events, the utilities, reward and costs are exposed. Afterwards, the dynamic programming approaches for solving

MDPs, as well as the limitations of the original MDP formulation are detailed.

**States and State Transition** A *state* is a description of the system at a particular point in time (Boutilier et al., 1999a). However, it is common to assume that the state captures all information relevant to the agent's decision-making process. For our problem, we assume a finite *state space* $S = \{s_1, ..., s_N\}$ of possible system states.

A discrete-time stochastic dynamic system consists of a state space and probability distributions governing possible *state transitions*, i.e. how the *next state* of the system depends on past states. These distributions constitute a model of how the system evolves over time in response to actions and exogenous events, reflecting the fact that the effects of actions and events may not be perfectly predictable even if the prevailing state is known.

States should be thought of as descriptions of the system being modeled, so the question arises of how much detail about the system is captured in a state description. More detail implies more information about the system, which in turn often translates into better predictions of future behavior. Of course, more detail also implies a larger set $S$, which can increase the computational cost of decision making.

It is commonly assumed that a state contains enough information to predict the next state. In other words, any information about the *history* of the system relevant to predicting its future is captured explicitly in a state itself. Formally, this assumption, *the Markov assumption*, says that knowledge of the present state renders information about the past irrelevant to making predictions about the future:

$$P(s'|s_0, s_1, ..., s_n) = P(s'|s_n) \tag{3.1}$$

The left member of this equation denotes the probability ($P$) of being in a state $s'$ depending of the history, starting in state $s_0$ and ending at the actual state $s_n$. The equations states that state $s'$ depends only on state $s_n$ and not on all the history.

In this thesis, we generally restrict our attention to finite-state, stochastic dynamical systems with the Markov assumption, commonly called Markov chains. Markovian models can be represented graphically using a structure as depicted in Figure 3.1 which reflects the fact that the current state is sufficient to predict future state evolution.

**Actions** A key element to MDPs is the set of actions available to the decision maker. When an action is performed in a particular state, this state changes stochastically in response to the action. We assume that the agent takes some action at each stage of the process, and then the system changes state accordingly.

For each state $s$, the agent has an available set of actions $A(s)$. This is called the *feasible set* for $s$. To describe the effects of $a \in A(s)$, we must supply the state-

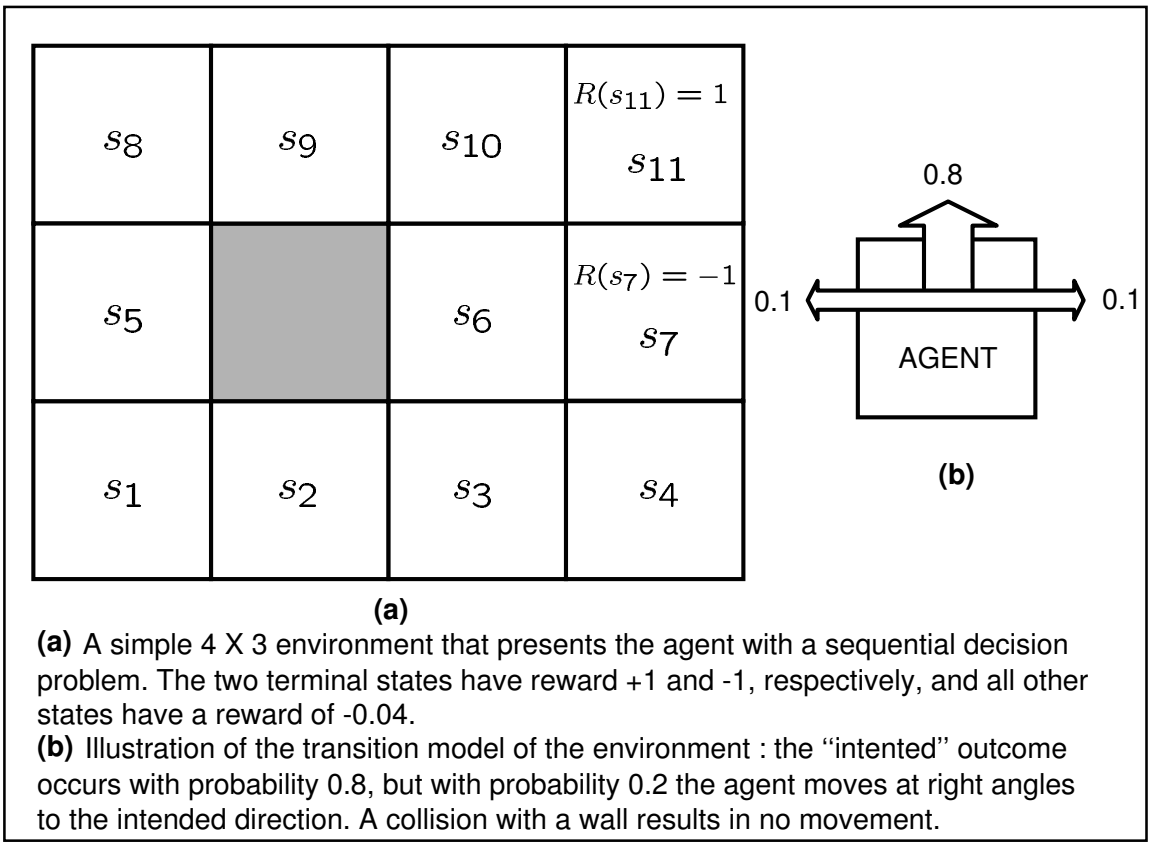
transition distribution $P_a(s'|s)$ for all actions $a$, states $s$. The terms $P_a(s'|s)$ are a family of distributions parameterized by $S$ and $A(s)$.

**Exogenous Events** *Exogenous Events* are those events that stochastically cause state transitions, much like actions, but beyond the control of the decision maker or agent. These might correspond to the evolution of a natural process or the action of another agent. We have two ways to model exogenous events. Firstly, we can use an *implicit-event model*, where the effects of the exogenous event are folded into the transition probabilities associated with the action. We often think of transitions as determined by the effects of the agent's chosen action and those of certain *exogenous events* beyond the agent's control, each of which may occur with a certain probability. When the effects of actions are decomposed in this fashion, we call the action model an *explicit-event model*.

**Values, Rewards and Costs** To decide which action to select at each stage, the agent needs some way to judge the *quality* of a course of action. An agent act in each state by maximizing its expected reward $R(s)$ and/or minimizing its expected cost $C(a, s)$. This is done by defining a value function $V(s)$ for each state. This value function depends on an environment history $h$ rather than on a single state. Roughly speaking, the value of a state is the expected value of the state sequences that might follow it until a goal state is reached.

The first question to answer is whether there is a *finite horizon* or an *infinite horizon* value for decision making. A finite horizon value means that there is a fixed time $N$ after which nothing matters — the game is overs, so to speak. Thus $V_h([s_0, s_1, ..., s_{N+k}]) = U_h([s_0, s_1, ..., s_N])$ for all $k > 0$. For example, suppose an agent starts at $s_3$ in the $4 \times 3$ world of Figure 3.2, and suppose $N = 3$. Then, to have any chance of reaching the $+1$ state, the agent must head directly for it, and the optimal action is to go $Up$. On the other hand, if $N = 100$ then there is plenty of time to take a safe route by going $Left$. So, with a finite horizon, the optimal action in a given state could change over time. We say that the optimal policy for a finite horizon is *nonstationary*. With no fixed time limit, on the other hand, there is no reason to behave differently in the same state at different times. Hence, the optimal action depends only on the current state, and the optimal policy is *stationary*. Policies for infinite-horizon case are therefore simpler than those for the finite-horizon case. Note that "infinite-horizon" does not necessarily mean that all state sequences are infinite; it just means that there is no fixed deadline. In particular, there can be finite state sequence in an infinite-horizon MDP containing a terminal state.

The next question to answer is to decide how to calculate the value of state sequence. It turns out that under stationarity there are just two ways to assign utilities




**(a)** A simple 4 X 3 environment that presents the agent with a sequential decision problem. The two terminal states have reward +1 and -1, respectively, and all other states have a reward of -0.04.
**(b)** Illustration of the transition model of the environment : the "intented" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement.

Figure 3.2: An agent willing to maximize its expected reward (from Russell and Norvig (2003)).

to sequences:

1. *Additive rewards*: The value of a state sequence is:

$$U_h([s_0, s_1, s_2, ...]) = r(s_0) + r(s_1) + r(s_2) + ...$$

2. *Discounted reward*: The value of a state sequence is:

$$U_h([s_0, s_1, s_2, ...]) = r(s_0) + \gamma \times r(s_1) + \gamma^2 \times r(s_2) + ...$$

where the discount factor $\gamma$ is a real number between 0 and 1. The discount factor describes the preference of an agent for current rewards over future rewards. When $\gamma$ is close to 0, rewards in the distant future are viewed as insignificant. When $\gamma$ is 1, discounted rewards are exactly equivalent to additive rewards, so additive rewards are a special case of discounted rewards. Discounting appears to be a good model of both animal and human preferences over time.

If the environment contains terminal states and if the agent is guaranteed to get one eventually, then we can use the infinite horizon with additive rewards criterion. Indeed, a policy that is guaranteed to reach a terminal state is called a *proper policy*. The existence of improper policies can cause the standard algorithms for solving MDPs to fail with additive rewards, and so provide a good reason for using discounted rewards. The next section details dynamic programming which is usually used to solve an MDP.

**Dynamic Programming Approaches** Suppose that an MDP is given with a state space $S$, action space $A$, a transition matrix $P_a(s'|s)$ for each action $a$, a reward function $R$, and a cost function $C$. The main problem is to find the policy that maximizes the expected total reward for the planning horizon. An example of a policy is given in Figure 3.3. Bellman's *principle of optimality* (Bellman, 1957) forms the basis of the stochastic dynamic programming algorithms used to solve MDPs. In particular, the optimal value of a state is the immediate reward for that state plus the expected discounted value of the next state transition probability, assuming that the agent chooses the optimal action. That is, the value of a state when its expected rewards are maximized is given by Russell and Norvig (2003).

$$V(s) = R(s) + \max_{a \in A(s)} \gamma \sum_{s' \in S} P_a(s'|s) V(s') \tag{3.2}$$

In the case where one reason on costs, the value of a state when its expected costs are minimized is given by Bonet and Geffner (2001).

$$V(s) = \min_{a \in A(s)} [C(a, s) + \gamma \sum_{s' \in S} P_a(s'|s) V(s')] \tag{3.3}$$

Finally, with costs and rewards, the following equation holds (Boutilier et al., 1999a)

$$V(s) = R(s) + \max_{a \in A(s)} [-(C(a,s)) + \gamma \sum_{s' \in S} P_a(s|s')V(s')] \tag{3.4}$$

A concept that is often useful is that of a Q-function (or Q-value). $Q(a,s)$ of each state action pair using the following equation:

$$Q(a,s) = R(s) + \gamma \sum_{s' \in S} P_a(s'|s) \max_{a' \in A(s')} Q(a',s') \tag{3.5}$$

where the optimal value of a state is $V^\star(s) = \max_{a \in A(s)} Q(a,s)$ Intuitively, $Q(a,s)$ denotes the expected value of performing action $a$ at states $s$ (Watkins and Dayan, 1992).

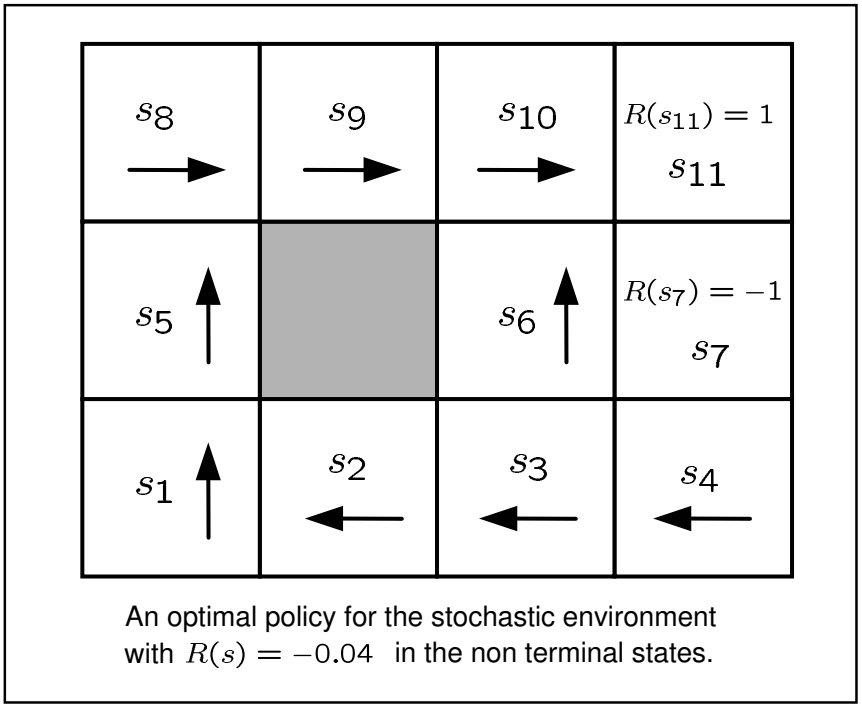



Figure 3.3: A policy for the example in Figure 3.2 (from Russell and Norvig (2003)).

Dynamic programming is said to be an *implicit-enumeration* approach because it finds an optimal solution, to a given problem, without evaluating all possible solutions. Once an optimal solution for a state is found, Bellman's principle of optimality allows us to infer that an optimal solution that reaches this state must include the solution that is optimal for this state. Using Bellman's principle of optimality to avoid enumerating all possible solutions is sometimes called *pruning by dominance*. In particular, the standards dynamic programming algorithms to solve an MDP are *value iteration* and *policy iteration*. We now detail these two types of iterations.

**Value Iteration** Equations 3.2, 3.3 and 3.4 form the basis of the value iteration algorithm for solving MDPs. If there are $n$ possible states, then there are $n$ Bellman equations, one for each state. The $n$ equations contain $n$ unknowns — the utilities of the states. So, we would like to solve these simultaneous equations to find the utilities. There is one problem: the equations are *nonlinear*, because the "max" operator is not a linear operator. Whereas systems of linear equations can be solved quickly using linear algebra techniques, systems of nonlinear equations are more problematic. One thing to try is an *iterative* approach. The initial utilities are selected arbitrary. Afterwards, the right-hand side of the equation is calculated, and plugged into the left-hand side — thereby updating the value of each state from utilities of its neighbors. This operation is repeated until an equilibrium is reached. Algorithm 3.1 taken from Russell and Norvig (2003) describes the value iteration algorithm.

**Algorithm 3.1** The value iteration algorithm for calculating utilities of states (Bellman, 1957).

**Function** VALUE-ITERATION($S$)
**returns** a value function
**repeat**
  $V \leftarrow V'$
  $\delta \leftarrow 0$
  **for all** state $s$ **in** $S$ **do**
    $V'(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P_a(s'|s) V(s')$
    **if** $|V'(s) - V(s)| > \delta$ **then**
      $\delta \leftarrow |V'(s) - V(s)|$
    **end if**
  **end for**
**until** $\delta < \epsilon$
**return** $V$

If this algorithm is applied infinitely to an MDP, an equilibrium is guaranteed of being reached, in which case the final utilities must be solutions to the Bellman equations. In fact, these are also the *unique* solutions, and the corresponding policy is optimal. This algorithm propagates utilities from a state $s$ to its neighbor states $s'$ iteratively. Indeed, value iteration can be viewed as *propagating information* through the state space by means of local updates. This algorithm terminates when the value change between two iterations is bounded by a constant $\epsilon$.

**Policy Iteration** Howard (1960)'s policy iteration algorithm is an alternative to value iteration for solving MDPs. Rather than iteratively improving the estimated

value function, it instead modifies the policies directly. It begins with an arbitrary policy $\pi$, then iterate, computing $\pi'$ from $\pi$. The policy iteration algorithm alternates the following two steps, beginning from some initial policy $\pi$:

- *Policy evaluation*: given a policy $\pi$, calculate $V = V^{\pi}$, the value of each state if $\pi$ were to be executed.
- *Policy improvement*: calculate a new policy $\pi'$, using one-step look-ahead based on $V$ (as in Equation 3.2).

The algorithm terminates when the policy improvement step yields no change in its utilities. At this point, we know that the value function $V$ is a fixed point of the Bellman update, so it is a solution to the Bellman equations, and $\pi$ must be an optimal policy. Because there are only finitely many policies for a finite state space, and each iteration can be shown to yield a better policy, policy iteration must terminate. Algorithm 3.2 taken from Russell and Norvig (2003) details the policy iteration algorithm.

**Algorithm 3.2** The policy iteration algorithm for calculating utilities of states (Howard, 1960).

**Function** POLICY-ITERATION($S$)
**returns** a policy
**repeat**
  **for all** state $s$ **in** $S$ **do**
    $V(s) \leftarrow R(s) + \gamma \sum_{s'} P_{\pi(s)}(s'|s)V(s')$
    {policy evaluation}
  **end for**
  $unchanged \leftarrow true$
  **for all** state $s$ **in** $S$ **do**
    **if** $\max_{a \in A(s)} \sum_{s'} P_a(s'|s)V(s') > \sum_{s'} P_{\pi(s)}(s'|s)V(s')$ **then**
      $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P_a(s'|s)V(s')$
      {policy improvement}
      $unchanged \leftarrow false$
    **end if**
  **end for**
**until** $unchanged \leftarrow true$
**return** $\pi$

The important point of the policy iteration algorithm is that the equation are *linear* for the policy evaluation part. Indeed, the "max" operator has been removed. For $n$

states, we have $n$ linear equation with $n$ unknowns for this part of the algorithm. This can be solved in time exactly $O(n^3)$ by standard linear algebra methods.

For small state spaces, policy evaluation using exact solution methods is often the most efficient approach. For large state space, $O(n^3)$ time might be prohibitive. Also, the policy improvement part of the algorithm is still nonlinear due to the presence of the "max" operator. Our concern is to use MDPs for the resource allocation of a ship which is a NP-Complete problem (Lloyd and Witsenhausen, 1986). The next section details the limitation of an MDP approach to our problem.

**Limitations of Original MDPs to Solve our Problem** In Computer Science, a polynomial algorithm is known to be effective. An exponential algorithm can cause problems, especially for real-time, because it could take a long time to obtain a solution. It is known that MDPs can be solved in polynomial time according to the number of states, using algorithms like value iteration or policy iteration. However, the complexity of general resource allocation problems leads to a large number of states (Lloyd and Witsenhausen, 1986). Indeed, AI planning systems that solve MDPs are faced with Bellman's so-called *curse of dimensionality* (Bellman, 1957): the number of states grows exponentially with the number of variables that characterize the planning domain. Consequently, a polynomial time algorithm is prohibitive for computing the policy of resource allocation problem.

Another major problem of MDPs is that the action space $|A|$ is can be very high. For a standard MDP approach, the number of actions to consider in a state is:

$$\prod_{i=1}^{nbAgents} nbChoices_i \tag{3.6}$$

where $nbChoices_i$ is the number of possible resource allocations for agent $i$, and $nbAgents$ is the number of agents. In this case, the value of all action combinations of each resource have to be computed. For example, if in a specific state, three resource types are available, with ten possible actions for each resource type, there are $10^3 = 1\ 000$ Q-values to compute for this state.

Thus the huge action and state spaces may be problematic to the attainment of a solution in due time for a resource allocation problem. The next section describes a resource allocation problem, which is very similar to our ship problematic. This simple example is described to understand all the aspects of our problematic.

### Decomposition Techniques

Another general method for tackling large stochastic problems is decomposition. A decomposition aims at reducing the planning time to generate a solution to the problem.

A decomposition can output an approximate or an optimal solution. If the decomposition may generate an optimal solution, the problem must have some specific domain dependent characteristics. The problem is either specified in terms of a set of "pseudo-independent" sub processes or automatically decomposed into such sub processes. These sub problems are then solved and the solutions to these sub problems are merged, or used to construct an approximate global solution. These techniques can be divided into two broad classes: those in which the state space of the problem is divided into regions to form sub problems, so that the problem is the union (in a loose sense) of the sub problems (Dean and Lin (1995); Precup and Sutton (1998); Laroche et al. (2001)); and those in which the sub problems are treated as concurrent processes, with their (loosely) cross-product forming the global problem (Singh and Cohn (1998); Boutilier et al. (1997); Meuleau et al. (1998); Wu and Castanon (2004); Russell and Zimdars (2003); Musliner et al. (2006)). Both these approaches offer great promise by allowing one to solve sub problems that are exponentially smaller than the global problem. If these solutions can be pieced together effectively, or used to guide the search for a global solution directly, dramatic improvements in the overall solution time can be obtained. The next section presents further simplifications to the problem model using characteristics to a resource allocation problem.

The important consideration in choosing a decomposition is that it is possible to represent each block compactly and to compute efficiently the consequences of moving from one block to another and, further, that the subproblems corresponding to the sub problems can themselves be solved efficiently. Chapter 6 presents three problem decomposition techniques for our resource allocation problem. The next two sections present the decomposition approaches by Meuleau et al. (1998) and Dolgov and Durfee (2004).

**Markov Task Sets** Meuleau et al. (1998) present an approach for solving MDPs where the problem is similar to ours. Furthermore, it was motivated by a military air campaign planning problem in which the tasks correspond to targets, and in which actions correspond to countering these target with weapons (missiles) transported in planes. There are global constraints on the total number of weapons available as well as instantaneous constraints (induced by the number of available planes) on the number of weapons that may be deployed on any single time step. Actions have inherently stochastic outcomes and the problem is fully observable. Thus, these problem characteristics match perfectly with the type of problem tackled in this thesis.

In this context, Meuleau et al. (1998) presented an approach for solving this problem using MDPs in which the subMDPs are treated as concurrent processes, with their (loosely) cross-product forming the global MDP. Indeed, the problem of sequential stochastic resource allocation is addressed. A number of different tasks, or objectives,

must be addressed and actions consist of assigning various resources at different times to each of these tasks. In addition, the state space of the MDP is assumed of forming a number of features that, apart from resources, are relevant only to a specific task. Furthermore, an assignment of resources to one task has no bearing on the features relevant to any other task. This means that each task can be viewed as an independent subprocess whose rewards and transitions are independent of the others, given a fixed action or policy (assignment of resources).

As a matter of fact, even this degree of independence does not generally make it easy to find an optimal policy. Resources are usually constrained, so the allocation of resources to one task at a given point in time restricts the actions available for others at every point in time. Thus, a complex optimization problem remains. If there are no resource constraints, the processes are completely independent. They can be solved individually, and an optimal global solution determined by concurrent execution of the optimal local policies; solution time is determined by the size of the subMDPs. With resource constraints, local optimal solutions can be computed, but merging them is now nontrivial. The question of how best to exploit local solutions to determine a global policy is the subject of the Meuleau et al. (1998) paper. They affirm that, for resource allocation problems, the action space is extremely large (every possible assignment of resources to tasks), making other standard approximation methods, such as neurodynamic programming (Bertsekas and Tsitsiklis (1996); Scherrer (2004)), unsuitable.

A hallmark of the heuristic algorithms described by Meuleau et al. (1998) is their division into two phases. An *off-line* phase computes the optimal solutions and value functions for the subMDPs associated with individual tasks. In an *on-line* phase, these value functions are used to compute a gradient for a heuristic search to assign resources to each task based on the current state. Once an action is taken, these resource assignments are reconsidered in light of the new state entered by the system. As for, Markov Task Sets (MTSs) is introduced by Meuleau et al. (1998) based on the MDP model.

A *Markov task set* (MTS) of $n$ tasks is defined by a tuple $\langle S, A, R, T, c, M_l, M_g \rangle$, where

- $S$ is a vector of state spaces, $s_1, ..., s_n$, where each $s_i$ is the set of primitive state(s) of Markov task $i$.
- $A$ is a vector of action spaces, $a_1, ..., a_n$, where each $a_i$ is a set of integers from 0 to some limit, describing the allocation of an amount of resource to task $i$.
- $R$ is a vector of reward functions $r_1, ..., r_n$, where $r_i : s_i \times a_i \times s'_i \times Time \rightarrow \mathbb{R}$, specifying the reward conditional on the starting state, resulting state and action at each time.

- $T$ is a vector of state transition distributions, $T_1, ..., T_n$, where $T_i : s_i \times a_i \times s_i' \rightarrow [0, 1]$, specifying the probability of a task entering a state given the previous state of the task and the action;
- $c$ is the cost for using a single unit of the resource.
- $M_l$ is the instantaneous (local) resource constraint on the amount of resource that may be used on a single step.
- $M_g$ is the global resource constraint on the amount of the resource that may be used in total.

The state space, for a finite planning horizon $H$, consists or the cross product of the individual state spaces and the available resources; the action space is the set of resource assignment, with an assignment being feasible at a state only if its sum exceeds neither $M_l$ nor $M_g$. Instead of formulating this "flat" MDP explicitly, the factored form is retained as much as possible. The goal, then, is to find an optimal non-stationary policy $\pi^* = \langle \pi^0, ..., \pi^{H-1} \rangle$, where $\pi^t = \langle \pi_0^t, ..., \pi_n^t \rangle$ and each $\pi_i^t : S \rightarrow A_i$ is a local policy for task $i$, that maximizes

$$E\left[\sum_{i=1}^{n} P_{pi_i}(s_i'|s_i) R_i(s_i, \pi_i(s)) - c \times \pi_i(s)\right] \tag{3.7}$$

subject to the constraints

$$\sum_{i=0}^{n} \pi_i(s) \leqslant M_l$$

and

$$\sum_{i=1}^{n} \pi_i(s) \leqslant M_g$$

Finding an optimal policy is a very hard problem even for small MTSs, because the equivalent MDP is very large. It is, for all practical purposes, impossible to solve exactly unless the number of tasks, the individual state spaces and the available resources are very small. The major source of difficulty is that the decision to apply a resource to a given task influences the availability of that resource (either now or in the future) for other tasks. Thus, the tasks, while exhibiting tremendous independence, still have strongly interacting solutions. A "local policy" for each task must take into account the state of each of the other tasks, precluding any state space reduction in general. Meuleau et al. (1998) their attention to approximation strategies that limit the scope of these interactions. They use an approximation strategy for MTSs called Markov Task Decomposition (MTD). The MTD method is divided into two phases. In the first, *off-line* phase, value functions are calculated for the individual tasks using dynamic

programming. In the second, *on-line* phase, these value functions are used to calculate the next action as a function of the current state of all processes.

The results of MTD shows that it can solve a problem in much less time than standard dynamic programming but with a little reduction in effectiveness. Indeed, the optimal Bellman equations indicate that an optimal allocation $a_i$ must not only take into account the future course of task $i$, but also reason about future contingencies regarding other tasks, and assess the value of reallocating some of these resources to other tasks in the future.

**Multiagent Operational Constrained MDP (M-OPER-CMDP)** At a high level, Dolgov and Durfee (2004) are interested in problems where the agents have a set of actions that are potentially executable, but each action requires a certain combination of resources. The amount of these shared resources is limited. Furthermore, each agent has constraints as to what resources it can make use of (for example, what equipment it can be outfitted with). In their model, execution begins with a distribution of the shared resources among the agents. Any resulting resource allocation must obey the constraints that no shared resource is over-utilized, i.e., the amount of all resources that are assigned to the agents does not exceed the total available amount. Furthermore, the assignment must satisfy the local constraints of the agents as to the resources that they can use. For example, it is useless (and thus essentially invalid) to assign to an agent more equipment than it can carry. Once the shared resources are distributed among the agents, they should use these resources to carry out their policies in such a way that the social welfare of the group (sum of individual rewards) is maximized. This technique can be modeled using Multiagent MDPs (MMDPs) (Boutilier, 1999).

In brief, the type of problem considered by Dolgov and Durfee (2004) is the same as the used in this thesis with one distinction: The allocation of resources are episodic, not sequential. Thus, the optimal policy contains decision to allocate the resources to a group of agents in a single stage, it does not consider reallocation. In particular, Dolgov and Durfee (2004) present a method that does not sacrifice optimality and, by fully exploiting the structure of the problem, makes it possible to solve problems orders of magnitude larger than what is possible using traditional multiagent MDP (Boutilier, 1999) techniques. Unlike the standard decomposition techniques, they do not divide the problem into subproblems and then recombine the solutions such as Meuleau et al. (1998) did. Instead, they formulate one policy optimization problem with constraints that ensure that the policies do not over-consume the shared resources.

More generally, it is often the case that an agent has many capabilities that are all in principle available to it, but not all combinations are realizable within the architectural limitations, because choosing to enable some of the capabilities might usurp resources needed to enable others. In other words, a particular policy might not be

*operational* because the agent's architecture does not support the combination of capabilities required for that policy. If this is the case, we say that the agent exhibits *operationalization* constraints.

Let us now consider a multiagent environment with a set of $n$ agents $M$, each of whom has its own set of states $S_m$ and actions $A_m$. In general, for a multiagent MDPs, we have to define a new state space that is the cross-product of the state spaces of all agents: $S(M) = S^n$, and a new action space that is the cross-product of the actions spaces of all agents: $A(M) = A^n$. The transition and reward functions are defined on the new state and action space, i.e., $P(M) : S^n \times A^n \times S^n \rightarrow [0, 1]$, and $R(M) : S^n \times A^n \rightarrow \mathbb{R}$. However, there is a large subclass of multiagent domains where the agents' rewards and transition functions are independent of each other, i.e., such problems are completely separable if there are no shared resources involved. The approach proposed by Dolgov and Durfee (2004) assume that once the shared resources are distributed, the agents operate completely independently of each other. In other words, each agent has its own independent reward and transition functions defined on $S$ and $A$.

Under the above independence assumptions, a joint policy of the group is simply the set of single-agent policies of all agents.

The multiagent policy optimization under limited shared resources problem can now be defined (Dolgov and Durfee, 2004). Let us say that there are several shared resources, and that every action of each agent requires some subset of these resources. Furthermore, all resources have costs associated with them and agents have upper bounds on the costs of resources that can be allocated to them. For example, a problem might involve shared equipment (e.g., tools) that enables agents to execute various actions, but each unit of equipment has some costs associated with it (e.g., weight), and the agents have upper bounds on how much weight they can carry. Under these conditions, the multiagent optimization constraint problem (M-OPER-CMDP) can be formulated as a tuple $\langle S, A, P, R, C, \hat{C}, Q, \hat{Q}, \alpha \rangle$ where:

- $S$ is a finite set of states.
- $A$ is a finite set of actions.
- $P^m : S \times A \times S \rightarrow [0, 1]$ defines the transition function for agent $m$. The probability that agent $m$ goes to state $s'$ if it executes action $a$ in state $s$ is $P_a^m(s'|s)$.
- $R^m : S \times A \rightarrow \mathbb{R}$ defines the rewards that agent $m$ receives. Agent $m$ gets a reward of $R_a^m(s)$ for executing action $a$ in state $s$.
- $C^m$, where $C_{ak}^m = \{0, 1\}$ defines action resource requirements. If agent $m$ needs resource $k$ to be able to execute action $a$, $C_{ak}^m = 1$; otherwise $C_{ak}^m = 0$.

- $\hat{C}$ defines the total amount of shared resources that are available to the group, i.e., there are $\hat{C}_k$ units of resource $k$ available to the agents.
- $Q$ defines the costs (weight, money, etc.) of each resource. The cost of type $l$ of a unit of resource $k$ is given by $Q_{kl}$.
- $\hat{Q}^m$ defines the upper bounds on how much of the costs the agent can incur (e.g., how much weight the agent can hold or how much money it can spend). Agent $m$ cannot exceed $\hat{Q}^m_l$ units of cost of type $l$.
- $\alpha^m$ is the initial probability distribution. The probability that agent $m$ starts in state $i$ is $\alpha^m_i$.

Dolgov and Durfee (2004) assume that the action space $A$ and the state space $S$ are the same for all agents. The goal of the optimization problem is to find a joint policy $\pi(M)$ that yields the highest expected reward, under the conditions that the shared resources are not over-utilized, and that no agent is assigned more resources than it can hold. In other words, we have to solve the following (abstract) math program:

$$\max V(\pi, \alpha) \left| \begin{array}{l} \sum\limits_m \theta \left( \sum\limits_a C^m_{ak} \sum\limits_s \pi^m_{ia} \right) \leqslant \hat{C}_k, \\ \sum\limits_k Q_{kl} \theta \left( \sum\limits_a C^m_{ak} \sum\limits_s \pi^m_{ia} \right) \leqslant \hat{Q}^m_l \end{array} \right. \tag{3.8}$$

where $\theta$ is a "step" function of a non-negative argument, defined as:

$$\theta(z) = \begin{cases} 0 \text{ if } z = 0 \\ 1 \text{ if } z > 0 \end{cases}$$

The first constraint in Equation 3.8 means that the total amounts of resources that are needed by all agents do not exceed the total amounts that are available. Indeed, $\sum_s \pi^m_{ia}$ is greater than zero only if agent $m$ plans to use action $a$ with nonzero probability. Thus, $\sum_a C^m_{ak} \sum_s \pi^m_{ia}$ is greater than zero when the agent plans to use actions that require resource $k$, in which case:

$$\theta \left( \sum_a C^m_{ak} \sum_s \pi^m_{ia} \right) = 1,$$

, and the first summation over all agents $m$ gives the total requirements for resource $k$, which should not exceed $\hat{C}_k$. The second constraint is analogous to the first one and has the meaning that the cost of type $l$ of the resources assigned to agent $m$ does not exceed its cost bounds $\hat{Q}^m_l$.

Dolgov and Durfee (2004) solves this math problem using a mixed integer linear program which allows to make use of a variety of highly optimized algorithms and tools for solving integer programs. Their approach offers an significant reduction of

the time to produce a solution compared to the traditional multiagent MDP approach. Dolgov and Durfee (2004) propose that their work could be greatly extended by combining it with an abstraction approach such as the one proposed by Boutilier et al. (2000). However, the authors state that it would involve overcoming several challenges, the most important of which is probably the following. Just like the majority of methods for solving unconstrained MDPs, the existing methods that work with compact problem representations rely on Bellman's principle of optimality, which states that the optimal action for each state is independent of the optimal actions chosen for other states. However, this principle no longer holds when global constraints are imposed on agents' policies. Indeed, enabling an optimal action for one state might consume limited resources, making the optimal action for another state infeasible. Overcoming such difficulties in an attempt to combine compact MDP representations and a constrained optimization ideas is a great direction to work on.

Also, MDPs can be solved efficiently using heuristic search, which is described in detail in Section 5.1.

### Partially Observable Markov Decision Processes (POMDPs)

Partially Observable Markov Decision Processes (POMDPs) are closer than MDPs to the general model of a decision processes. POMDPs have generally been studied with the assumption of stationarity and optimality criteria identical to those of MDPs. In brief, a POMDP can be viewed as an MDP with a state space consisting of the set of probability distributions over $S$. These *probability distributions* represent states of belief: the agent can "observe" its state of belief about the system although it does not have exact knowledge of the system state itself.

POMDPs provide a natural and expressive framework for decision making, but their use in practice has been limited by the lack of scalable solution algorithms. Two important sources of intractability plague discrete model-based POMDPs: high dimensionality of belief space, and the complexity of policy or value function space. Classic solution algorithms (Cassandra et al. (1997); Kaelbling et al. (1998); Hansen (1998)), for example, compute value functions represented by exponentially many value vectors, each of exponential size, according to the number of states. As a result, they can only solve POMDPs with on the order of 100 states. Consequently, much research has been devoted to mitigating these two sources of intractability. The complexity of policy/value function space has been addressed by observing that there are often very good policies whose value functions are representable by a small number of vectors. Various algorithms such as approximate vector pruning (Hansen and Feng, 2001), point-based value iteration (PBVI) (Pineau et al. (2003); Spaan and Vlassis (2004)), bounded policy iteration (BPI) (Poupart and Boutilier, 2004), gradient ascent (GA) (Meuleau et al. (1999); Aberdeen and Baxter (2002)) and stochastic local

search (SLS) (Braziunas and Boutillier, 2004) exploit this fact to produce (often near-optimal) policies of low complexity (i.e., few vectors) allowing larger POMDPs to be solved. Still these scale to problems of only roughly 1000 states, since each value vector may still have exponential dimensionality. Conversely, it has been observed that belief states often carry more information than necessary. Hence, one can often reduce vector dimensionality by using compact representations such as decision trees (DTs) (Boutilier and Goldszmidt, 1996), algebraic decision diagrams (ADDs) (Hansen and Feng, 2001), or linear combinations of small basis functions (LCBFs) (Guestrin et al., 2001), or by indirectly compressing the belief space into a small subspace by a value-directed compression (VDC) (Poupart and Boutilier, 2003) or exponential PCA (Roy and Gordon, 2002). Once compressed, classic solution methods can be used. However, since none of these approaches address the exponential complexity of policy/VF space, they can only solve slightly larger POMDPs (up to 8250 states (Roy and Gordon, 2002)). Scalable POMDP algorithms can only be realized when both sources of intractability are tackled simultaneously. While Hansen and Feng (2001) implemented such an algorithm by combining approximate state abstraction with approximate vector pruning, they didn't demonstrate the scalability of the approach on large problems. Furthermore, Poupart and Boutilier (2004) describe how to combine value directed compression (VDC) with bounded policy iteration (BPI).

They demonstrate the scalability of the resulting algorithm (VDCBPI) on synthetic network management problems of up to 33 million states. Among the techniques that deal with the curse of dimensionality, VDC offers the advantage that the compressed POMDP can be directly fed into existing POMDP algorithms with no (or only slight) adjustments. This is not the case for exponential-PCA, nor compact representations (DTs, ADDs, LCBFs). Glazebrook and Washburn (2004) overview works on "shoot-look-shoot-" strategies in the presence of imperfect target status knowledge. The finite horizon, which is the case where the number of weapon is constrained is not even tackled their paper. Thus, much work has to be done in this field. Indeed, we would have to improve this research field to efficiently use POMDPs for our complex real-time problem. This open the door for many research avenues, such as learning, approximation, heuristic search, and so on. The next section introduces planning problems from the artificial intelligence domain.

### 3.1.2 Planning Problem in the AI Tradition

#### Deterministic Planning

The classical AI planning model assumes deterministic actions: any action $a$ taken at any state $s$ has at most one successor $s'$. The other important assumptions are non-observability and that value is determined by reaching a *goal state*: any plan that

leads to a goal state is preferred to any that does not. Often there is a preference for shorter plans; this can be represented by using a discount factor to "encourage" faster goal achievement or by assigning a cost to actions. Reward is associated only with transitions to goal states, which are absorbing[1]. Action costs are typically ignored, except as noted above.

In classical planning models it is usually assumed that the initial state is known with certainty. This contrasts with the general specification of MDPs above, which does not assume knowledge of or even distributional information about the initial state. Knowledge of the initial state and determinism allow optimal straight-line plans to be constructed, with no loss in value associated with non-observability, but unpredictable exogenous events and uncertain action effects cannot be modeled consistently if these assumptions are adopted.

The optimal deterministic planning paradigm augment this formalism to generate a plan which maximizes (or minimizes) a certain objective function. Furthermore, the classical planning assumption of omniscience can be relaxed somewhat by allowing the state of some aspects of the world to be unknown. The agent is, thus, in a situation where it is certain that the system is one of a particular set of states, but does not know which one. Unknown truth values can be included in the initial state specification, and taking actions can cause a proposition to become unknown as well. Conditional deterministic planners were introduced to consider this problem (Pryor and Collins, 1993).

### Probabilistic Planning

A direct probabilistic extension of the classical planning problem can be stated as follows: take as input (a) a *probability distribution* over initial states, (b) stochastic actions, (c) a set of goal states, and (d) a probability success threshold $\tau$. The objective is to produce a plan that reaches any goal state with probability at least $\tau$, given the initial state distribution. No provision is made for execution-time observation, thus straight-line plans are the only form of policy possible. This is a restricted case of the infinite-horizon MDP problem, one in which actions incur no cost and goal states offer positive reward and are absorbing. It is also a special case in that the objective is to find a *satisfying*[2] policy rather than an optimal one.

Draper et al. (1994) have proposed an extension of the probabilistic planning problem in which actions provide feedback, using exactly the observation model used by an MDP. Again, the problem is posed as that of building a plan that leaves the system

[1]Once an agent enters a closed set or absorbing state, it remains there forever with probability 1 (Boutilier et al., 1999a).

[2]A satisfying policy is one which have a probability of success at least $\tau$ (Boutilier et al., 1999a)

in a goal state with sufficient probability. But a plan is no longer a simple sequence of actions — it can contain conditionals and loops whose execution depends on the observations generated by sensing actions. This problem is a restricted case of the general POMDP problem: absorbing goal states and cost-free actions are used, and the objective is to find any policy that leaves the system in a goal state with sufficient probability.

### Contingency Planning

Majercik and Littman (2003) makes a distinction between conditional planning and contingent planning. In conditional planning, the effects, but not the execution, of actions are contingent on the outcome of previous actions. In contingent planning, both the effects and execution of actions are contingent on the outcomes of previous actions. Thus, in contingent planning, the agent can make observations and construct a branching plan in which actions are made contingent on these observations. Without the ability to observe its environment and condition its actions accordingly, an agent can only execute a *straight-line plan*, a simple non-contingent sequence of actions, and hope for the best. Such a plan can also be called "open loop", in contrast to "closed loop" plans that condition action choices on run-time observations.

In previous work, a contingency planner using a greedy algorithm improved with a tabu search heuristic was considered (Plamondon (2003); Soucy (2003)). This approach produced good results in a short time but it has two main inconvenient. Firstly, when computing the solution, the algorithm does not know how far it is form the optimal one. Indeed, it does not even know if it as found an optimal solution. Also, no efficient representation of a sequential environment is made, which dynamic programming does successfully. Thus, when the planning horizon is high, the number of possible allocations is very high for tabu search.

## 3.2 Planning Approaches for Resource Allocation

The problem of resource allocation among multiple self-interested agents is ubiquitous, with applications ranging from decentralized scheduling (Wellman et al., 2001) and network routing (Feldmann et al., 2003) to transportation logistics (Sheffi (2004); Song and Regan (2002)) and bandwidth allocation (McMillan (1994); McAfee and McMillan (1996)), just to name a few. A lot of work in this setting has focused on the problem of mechanism design (Mas-Colell et al., 1995), the goal of which is to create mechanisms that allocate the resources to the agents in ways desirable to the system-designer, given that each participating agent is selfishly maximizing its own utility. In the next section, we introduce some general considerations on planning.

### 3.2.1 General Considerations on Planning for Resource Allocation

In general, there are two known approaches for allocating resources to tasks (Hosein and Athans, 1990). Firstly, the episodic approach which uses all the available resources simultaneously. Secondly, the sequential version, as used in this thesis, which extends the episodic version by having a number of stages in the scenario. In this last context, the planning process is allowed to observe the outcomes of all engagements of the previous stage before assigning and committing resources for the present stage. This approach uses fewer resources and has a higher expectation to achieve the tasks than the episodic approach.

Resource allocation problems are known to be NP-Complete (Zhang, 2002). Since resources are usually constrained, the allocation of resources to one task restricts the options available for other tasks. The complexity of this problem is exponential according to the number of resources and tasks, and many approximations and heuristics have been proposed (Meuleau et al. (1998); Wu and Castanon (2004); Aberdeen et al. (2004)). An effective approach is to plan for the resources separately as proposed by Wu and Castanon (2004). Wu and Castanon formulates a policy for each resource and a greedy global policy is produced by considering each resource in turn, producing an approximate policy. Their coordination rules are, however, sometimes very specific to the problem's characteristics.

Russell and Zimdars (2003) propose another *central* agent coordination scheme. Their Q-Decomposition Reinforcement Learning coordination process determines the Q-values which maximize the sum for each states at each learning iteration. An important assumption of this method is that each agent should have its own independent reward function. Thus, for a resource allocation problem, there would have an agent for each task to achieve.

Since resources have local and global resource constraints on the number that be used, the problem here can be viewed as a constrained Markov decision process (Bertsekas (2005); Feinberg and Shwartz (1996); Dolgov and Durfee (2004); Altman (1999)). In this context, dynamic programming (Bertsekas (2005); Feinberg and Shwartz (1996)) or linear programming (Dolgov and Durfee, 2004) may be used to obtain a policy.

Boukhtouta (2002) investigated which type of planning approach should be used in military operations. In his paper he compared two main approaches to solve a decision problem, the conventional planning and the Markov Decision Processes (MDPs) approaches. In a complex dynamic military environment, it could be difficult to elaborate an initial plan which contains all the response to every contingencies that could arise. Many planning paradigm used a replanning approach to overcome this prob-

lem. In particular, the conventional planning Operational Research (OR) approaches mainly use replanning-based methods to deal with uncertainty. Replanning approaches have been used in developing many military planners. DART (Dynamic Analysis and Replanning Tool), TARGET (Theatre-level Analysis, and Graphical Extension Toolbox)[3], and Cypress-SIPE2 (Wilkins et al., 1995) are military system planners that use the replanning approach. Moreover, many decision-support planners have been model as a Constraint Satisfaction Problem (CSP) (Zweben and Fox (1994); Mailler et al. (2003)). However, the naive constraint propagations methods are not formulated to handel stochastic problems. They have been modified for stochastic problems but it replans at every step, and its policy does not consider the fact that it will be able to replan in the future.

Ultimately, Boukhtouta concluded that conventional OR planning is insufficient to model subtlety many military planning problems. However, planning methods based on MDP or a POMDP can be used to develop more sophisticated planners. An MDP style approach seems more efficient than conventional planners. The MDP specifies a closed-loop policy that makes optimal decisions with full foresight of the remaining uncertainty of the military problem. Conventional planning methods based on replanning cannot provide such a solution. Although MDPs and POMDPs are promising tools to use in developing planners, they suffer from the so-called *curse of dimensionality* (Bellman, 1957): the number of states grows exponentially with the number of variables that characterize the planning domain. Consequently, a polynomial time algorithm can be prohibitive for a real-time application. However, many strategies can be adopted to overcome this (see Boutilier et al. (1999a) for a general overview).

In other related works, Chalmers (1995) presents a model of a deliberative planning function embedded in a real-time layered control architecture for a weapon engagement manager in a naval AWW system. This work aims at producing an efficient system for threat evaluation and weapon assignment. We should note that computing an optimal plan is NP-hard, which makes it impossible to guarantee optimal response within real-time deadlines. To overcome this, Chalmers (1995) outlined an adaptive approach based on knowledge compilation done a priori, and classification of battle scenarios done with an anytime planner which permits a resource-limited agent to generate effective plans. Missions goals may include: maximize threat value destroyed (subtractive defence), maximize value of surviving assets (preferential defence), and minimize wastage of missiles.

Design-to-time scheduling (Garvey et al. (1993); Garvey and Lesser (1993)) could be a viable approach to model our resource allocation problem. In brief, design-to-time scheduling includes a set of task to realize and a set of methods to realize these tasks. Each methods are associated a quality and a duration to execute each task. The term

[3]DART and TARGET are developed by BNN Technologies for the U.S. Department of Defence

quality summarizes several possible properties of actions or results in a real system: certainty, precision and completeness of a result. There are interactions among the tasks. For example, if the execution of task $ta_1$ is made before the execution of task $ta_2$, $ta_1$ may increase or decrease the qualities and durations of some methods associated to $ta_2$.

For example, for resource allocation of a ship to counter incoming missiles, the tasks are the missiles attacking ownship. Also, the set of methods associated to a task $ta$ are all the possible resource allocations which the ship can perform to execute $ta$. In its initial form, design-to-time scheduling was developed for deterministic and episodic problems. Since our problem is stochastic and sequential, the initial form of design-to-time scheduling cannot be used as a modeling tool in this thesis.

A good recent extension of design-to-time scheduling was proposed by Ramsauer (2002). This extension considers that each action can have an uncertain ending time. In particular, he has discretized the ending time of an action in different possible ending times. The more the action space is discretized, the more the branching factor and the computing time to generate a solution augments. We will talk in Section 7.7.3 that discretizing the action time can be very useful in our works.

A major drawback of design-to-time scheduling is its ability to represent effectively a stochastic environment. Indeed, by accumulating tasks qualities using maximum and minimum functions, the resulting root task quality does not represent an expectation of the number of tasks that are going to be realized, like MDPs do. For this reason, design-to-time scheduling does not consider well the case where multiple methods can be used simultaneously to execute a task, where each method has a certain probability to execute the task.

On the other hand, in an MDP, the optimal value represents the reward an agent is expected to obtain by executing the current policy. We think an MDP matches more with our problem than design-to-time scheduling for this reason. However, when merged with MDPs, as done by Musliner et al. (2006) the resulting framework seems pretty effective. But, the complexity of the framework is the same as using MDPs only. Thus, in our opinion, MDPs is a more viable tool than design-to-time scheduling to model a stochastic resource allocation problem.

Furthermore, a rollout algorithm (Bertsekas, 2005), which is an approximation of dynamic programming, could be used to model our problem. In this algorithm, the value of the states are initialized using a lower bound heuristic and a backup is performed on reachable states until a certain depth is reached. The value of the further states can be approximated using monte-carlo simulation. We further discuss the rollout algorithm in Section 5.2.7.

## 3.3 Conclusion

In this context, a literature review for the specific problem of task planning under uncertainty has been given. Firstly, we have detailed the main planning approaches in the operation research and artificial intelligence tradition. Then, specific planning approaches for naval environments are detailed. In this context, the Weapon-Target Assignment (WTA) problem, the coordination problem and the movement problem are described.

The next chapter describes a very simple resource allocation problem to better understand the type of problem we are interested in. We also model this problem with Markov Decision Processes.

# Chapter 4

# Command and Control Systems: The Resource Allocation Problem

## 4.1 Introduction

In Section 3.1, we said that most problems lies between planning and scheduling. Scheduling, introduced in Chapter 2, is efficient in problems where there are constraints and resources, while planning, described in Chapter 3, is efficient in problems where there are a high number of possible actions. In this section, we present a problem which has constraint on resources and a high number of possible actions, thus requiring both planning and scheduling.

Our problem of interest, described in Section 4.3.2, are maritime environments which are known to be very complex environments with tight real-time constraints. In case of an own platform attack, the commander must make fast decisions by considering several factors to ensure himself of the best possible survival of the own platform and its crew. Under such real-time constraints, it can often happen that the commander makes errors because of the complexity of the environment or the stress which the situation can generate. In these conditions, a computer is tremendously faster than a human and consequently, it can suggest decisions in time thus facilitating the task of a commander.

As demonstrated in Section 4.3.2, our problem of interest falls into the *partially observable*, *stochastic*, *sequential*, *dynamic*, *discrete*, and *multiagent* case. However, our problem of interest is very complex and modeling a problem with all its subtleties would be very time consuming. We designed a simplified problem version which permit us to concentrate our energy on specific parts of the problem. This simplified problem is presented in Section 4.7.

## 4.2 Problem Example

A simple resource allocation problem is one where there are the following two tasks to realize: $ta_1 = \{\text{wash the dishes}\}$, and $ta_2 = \{\text{clean the floor}\}$. These two tasks are either in the realized state, or not realized state. To realize the tasks, two type of resources are assumed: $res_1 = \{\text{brush}\}$, and $res_2 = \{\text{detergent}\}$. A computer has to compute the optimal allocation of these resources to cleaner robots to realize their tasks. In this problem, a state represents a conjunction of the particular state of each task, and the available resources. The resources may be constrained by the amount that may be used simultaneously (local constraint), and in total (global constraint). Furthermore, the higher is the number of resources allocated to realize a task, the higher is the expectation of realizing the task. For this reason, when the specific states of the tasks change, or when the number of available resources changes, the value of this state may change.

When executing an action $a$ in state $s$, the specific states of the tasks change stochastically, and the remaining resource are determined with the resource available in $s$, subtracted from the resources used by action $a$, if the resource is consumable. Indeed, our model may consider *consumable* and *non-consumable* resource types. A consumable resource type is one where the amount of available resource is decreased when it is used. On the other hand, a non-consumable resource type is one where the amount of available resource is unchanged when it is used. For example, a brush is a non-consumable resource, while the detergent is a consumable resource.

The system is in a state $s$ with a set of task $Ta$ to realize, and a set $Res$ of resource available. A possible action in this state may be to allocate one unit of detergent to task $ta_1$, and one brush to task $ta_2$. The state of the system changes stochastically, as each task's state does. For example, the floor may be clean or not with a certain probability, after having allocated the brush to clean it. In this example, the state of the tasks may change, for example, in $n$ new possible combinations. For all these $n$ possible state transitions after $s$, the consumable resources available ($Res_c$) are $Res_c \setminus res(a)$, where $res(a)$ is the consumable resources used by action $a$. We now introduce Command and Control Systems which is the general problem which lead to this thesis.

## 4.3 Command and Control (C2) Systems

### 4.3.1 Overview

The increasing difficulty and diversity of open-ocean and littoral (i.e. near land) scenarios, and the volume of data of imperfect nature to be processed under time-critical

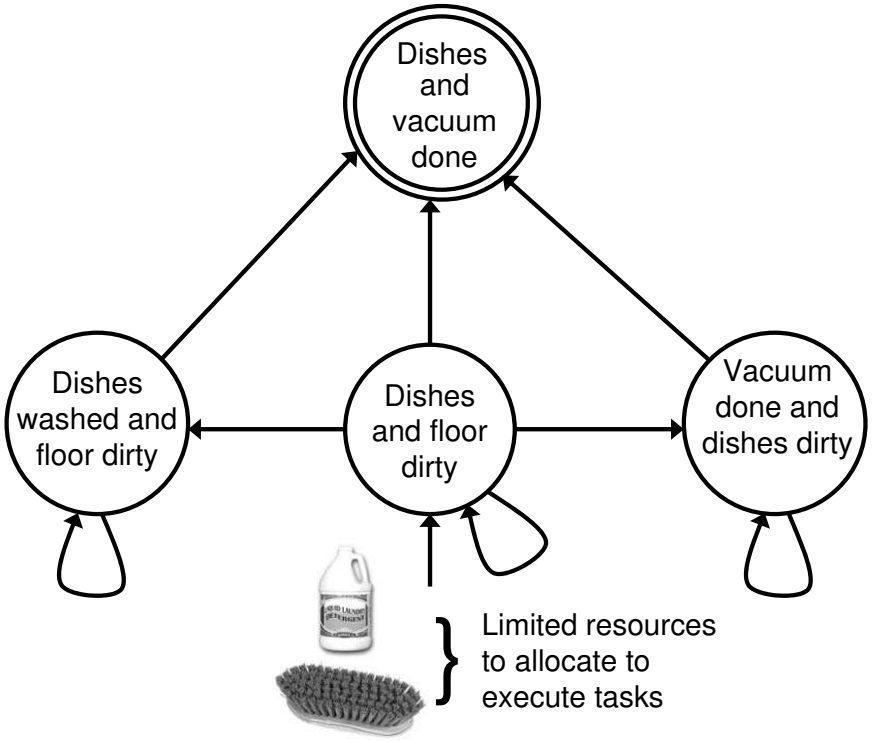



Figure 4.1: Task transition graph.

$$s(Ta, Res)$$
$$a(s) = \langle detergent_{ta_1}, brush_{ta_2} \rangle$$
$$s_1(Ta_1, Res \setminus \{a\}) \ldots \; s_n(Ta_n, Res \setminus \{a\})$$

Figure 4.2: State transition graph.

conditions pose significant challenges for future shipboard command and control systems and the combat system operators who must use these systems to defend their ship and fulfil their mission (Chalmers, 1997). For this thesis, we see the command and control system as a subsystem at the heart of a ship's combat system which includes various other subsystems like weapon and sensor systems, a navigation system, and an environment monitoring system. The command and control system provides automated capabilities to allow operators to use the fighting resources of a ship. However, current operational systems generally provide little support for tactical decision making in complex, highly dynamic scenarios where time for decision making and action execution is at premium. The need for such support is very pressing given the current emphasis on littoral warfare that results in reduced reaction times and the need to deal quickly and correctly with complex rules of engagements designed to increase efficiency of fighting resources and avoid undesirable consequences (Liang, 1995).

The Defence R&D Canada DRDC-Valcartier team, with their collaborators, have for several years now been investigating methods to augment or enhance existing command and control system capabilities— by continuously fusing data from a ship's sensors and other sources, dynamically maintaining a tactical picture, and supporting response to actual or anticipated threats.

DRDC (http://www.drdc-rddc.dnd.ca/) has been created in 1945. Its mission is to improve Canada's defence capability through research and development, e.g. expert counselling, development of improved operational capacities, monitoring of new technological developments, and technology transfer. Now a mainstay of Defence R&D Canada — an agency of the department of national defence — the center has resolutely brought itself into line with the new economic realities. It has abandoned its former organizational structure and embraced a new, more promising modus operandi. The emphasis is now on partnership with industry and universities. DRDC-Valcartier's scientists are striving to advance defence technology, to extend the limits of knowledge, and to exploit the economic opportunities created by its investment of time and energy. In this way, technologies and processes developed by those scientists are more easily carried through to application for non-military uses.

### 4.3.2 Major Considerations

Command and Control (C2) is the exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of a mission. C2 functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of a mission. C2 tasks usually include weapon and sensor systems control, tactical picture,

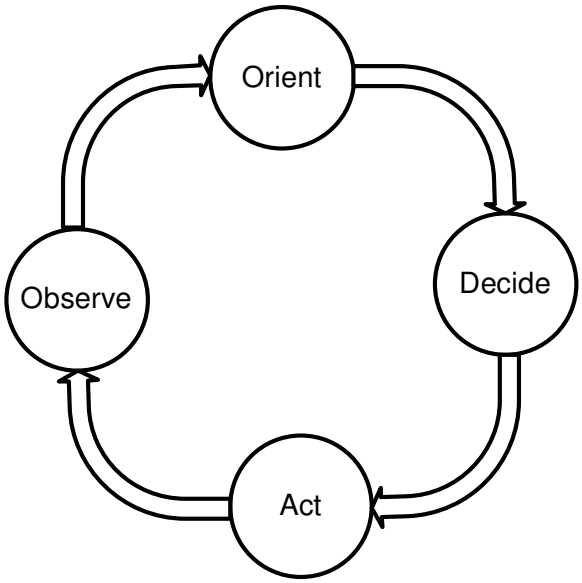



Figure 4.3: The OODA loop (Boyd, 1987).

compilation, situation interpretation and threat evaluation, weapon selection, engagement monitoring and mission planning and evaluation. In fact, C2 tasks cover what is called the OODA (Observe, Orient, Decide and Act) loop as depicted in Figure 4.3. This theory, by Boyd (1987), essentially states that, in a confrontation, whoever is quicker to react to changes will prevail. Boyd describes this readiness to react to changes as "having a tight OODA loop". Therefore, Boyd's theory can be expressed as "whoever has a tighter OODA loop, will prevail". Thus, the C2 process necessitates a highly dynamic flow of information. Decision-making involves a number of operators and sophisticated decision support systems with a concomitant requirement for developing a common, shared representation of the situation. In this context, there are many generic issues worth considering. Indeed, the development of a relevant C2 theory will have significant impact upon the analysis and design of both military and civil C2 systems. The major considerations are the following (Chaib-draa et al., 2001):

- *C2 is a multiagent environment:* A C2 system is a multiagent organization in which the decision-makers are both human and artificial agents. The decision-makers are often geographically scattered due to the operational environment and the physical nature of sensors and resources. Cooperation, coordination and communication between the decision-makers are thus critical in such a distributed C2 architecture.

- *C2 has a functional architecture:* Another key element of the C2 process is its functional decomposition. Indeed, the C2 process can be decomposed into a set of generally accepted C2 functions that must be executed in a reasonable time frame to ensure success.

- *C2 is a complex process:* The complexity of most C2 problems rises from the multitude, the heterogeneity and the interrelations of the resources involved. Generally, no decider alone can deal with the inherent complexity of the global situation. This leads to a decomposition of the decision process along distinct expertise dimensions. In light of these considerations, team training is essential in any C2 organisation to achieve superior coordination and to make the best utilisation of common resources. Moreover, a military C2 system must take into account the specific established command and decision hierarchy.

- *C2 deals with large volumes of data under stringent time constraints:* Perceptual and cognitive processing is further complicated by the fact that the underlying information is derived by continuously integrating and merging data from a variety of sources to build a coherent situational picture. Particular processing problems arise from information with different accurateness and timeliness. The integrated data is generally imperfect; it can be uncertain, incomplete, imprecise, inconsistent and ambiguous, due to limited sensor coverage, report ambiguities, report conflicts or inaccuracies in measured data (Waltz and Buede, 1986). It follows that (i) operators may have to handle potentially large situation uncertainties and, (ii) at any given moment, there may be several likely interpretations of the tactical picture. This leads to processing large volumes of data under stringent time constraints.

In the case of Above-Water Warfare (AWW), the list of functions of the C2 architecture is as follows:

1. *Threat detection*: Based on data from several sensors.
2. *Target tracking*: Usually based on data fusion.
3. *Discrimination*: Results in the resolution of true threats from decoys.
4. *Identification*: In this step, the threats are identified.
5. *Battle planning*: In this process, decisions are made on how to deal with the identified threats.
6. *Resource assignments*: Resources are assigned to engage each threat.
7. *Engagement control*: The process by which decisions in the two preceding steps are executed in real-time.
8. *Damage assessment*: This process evaluates the outcome of the engagement control.

In our concern, we address how we develop a decision support system that focusses specifically on some particular aspects of the C2, in order to reduce the complexity of the domain. Our primary focus is on the *battle planning*, *resources assignments* and *engagement control* processes. To this end, we consider the Situation and Threat Assessment (STA), which consists of *threat detection*, *target tracking*, *discrimination* and *identification*, as a black box. Therefore, the output of these steps is taken directly as available data, since it is not in the scope of this thesis to work on the STA. The *damage assessment*, which covers the evaluation of the damage to a ship, is not currently in our research goals, since we only need to evaluate the damages in a simple fashion: destroyed, damaged or intact. Not working on STA and damage assessment reduces the large volume of data that needs to be processed, which helps reducing the system's complexity.

### The Naval Environment

The complexity of military decision-making processes can be affected by a number of different environment properties. Russell and Norvig (2003) suggest the following classification of environment properties:

- *Fully observable* vs. *partially observable*: If an agent's sensor gives an access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. A task environment is effectively fully observable if the sensors detect all aspects that are *relevant* to the choice of action; relevance, in turn, depends on the performance measure. Fully observable environments are convenient because the agent does not need maintain any internal state to keep track of the world. An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data — for example, the sensors of a military platform may fail to detect the characteristics of a threat properly.

- *Deterministic* vs. *stochastic*: If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise, it is stochastic. In principle, an agent need not worry about uncertainty in a fully observable, deterministic environment. If the environment is partially observable, however, then it could *appear* to be perceived as stochastic. This is particularly true if the environment is complex, making it hard to keep track of all the unobserved aspects. Thus, it is often better to think of an environment as deterministic or stochastic *from the point of view of the agent*. A resource allocation system for a ship is stochastic since it cannot predict exactly what is the evolution of the environment according to the present state (for example: new threats emerging, threats trajectories

changing, own resource effects etc.).

- *Episodic* vs. *sequential*: In an episodic task environment, the agent's experience is divided into atomic episodes. Each episode consists of the agent perceiving and then performing a single action. Crucially, the next episode does not depend on the actions taken in previous episodes. In episodic environments, the choice of actions in each episode depends only on the episode itself. Many classification tasks are episodic. For example, an agent that has to spot defective parts on an assembly line, bases each decision on the current part, regardless of previous decision; moreover, the decision does not affect whether the next part is defective or not. In sequential environments, on the other hand, the current decision could affect all future decisions. Resource allocation for a ship is sequential in the sense that committing a given resource against a threat can have an impact on the availability of that resource for subsequent engagements. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

- *Static* vs. *dynamic*: If the environment can change while an agent is deliberating, then we say the environment is dynamic; otherwise, it is static. Static environments are easy to deal with because the agent does not need to worry about time. Dynamic environments, on the other hand, require continuous action from the agent. In this context, no action is interpreted as the action of doing no thing. If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is *semidynamic*. Resource allocation for a ship is dynamic because, the threats keep moving, changing directions, appearing, disappearing, etc. while the agent is planning.

- *Discrete* vs. *continuous*: The discrete/continuous distinction can be applied to the state of the environment, to the way time is handled, and to the percepts and actions of the agent. For example, a discrete-state environment such as a chess game has a finite number of distinct states. Chess also has a discrete set of percepts and actions. A ship environment can also being modeled discretely since the behavior of all threats and of a ship itself sweep into a finite number of distinct states.

- *Single agent* vs. *multiagent*: The distinction between single-agent and multiagent environments may seem simple enough. For example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two-agent environment. There are, however some subtle issues; we have to explain which entities *must* be viewed as agents. Does an agent $A$ (a bus driver for example) have to treat an object $B$ (another vehicle) as an agent, or can it be treated merely as a stochastically behaving object, analogous

to waves at the beach or leaves blowing in the wind? The key distinction is whether $B$'s behavior is best described as maximizing a performance measure whose value depends on agent $A$'s behavior. For example, in chess, the opponent entity $B$ is trying to maximize its performance measure, which, by the rules of chess, minimizes agent $A$'s performance measure. Thus, chess is a *competitive* multiagent environment. In the bus-driving environment, on the other hand, avoiding collisions maximizes the performance measure of all agents, so it is a partially *cooperative* multiagent environment. It is also partially because, for example, only one bus can occupy a bus stop. The agent-design problems arising in multiagent environments are often quite different from those in single-agent environment; for example, *communication* often emerges as a rational behavior in multiagent environments; in some partially observable competitive environments, we must use a *stochastic behavior* because it avoids the pitfalls of predictability. Resource allocation for a ship can be viewed as a competitive and cooperative multiagent environment. Indeed, a ship agent is in competition with 0 to $n$ threats attacking it. Also, it can cooperate with 0 to $k$ other ships to counter some threats.

As demonstrated, the naval above-water warfare resource management problem for a ship falls into the *partially observable*, *stochastic*, *sequential*, *dynamic*, *discrete*, and *multiagent* case.

The next section introduces the specific problematic of above-water warfare we are interested into. This problematic features the three parts of the decision process described afterwards: Weapon-target allocation, resource coordination, and movement.

### Target Problem

The resource management (weapons, navigation etc.) of a ship constitutes a real-time planning problem with very constraining temporal requirements. During the last years, RDDC-Valcartier undertook various research activities having as objectives the maximization of defensive capacities of a ship by optimizing the management and the deployment of its various resources. These resources were, however, considered separately. For example, the problems of resources allocation of a ship were always treated without taking movements of such ship into account, and vice versa. In this section, we consider an integrated approach, whose central element is the positioning and the operations of the ship. We now introduce different attacking phases for a threat which are related to different defence strategies. A threat attacking a ship has four phases. These four different phases, represented in Figure 4.4, correspond to different planning tactics for a ship : (1) platform search; (2) tracking by platform; (3) missile search; and (4) the locked phase. We now detail these different phases.

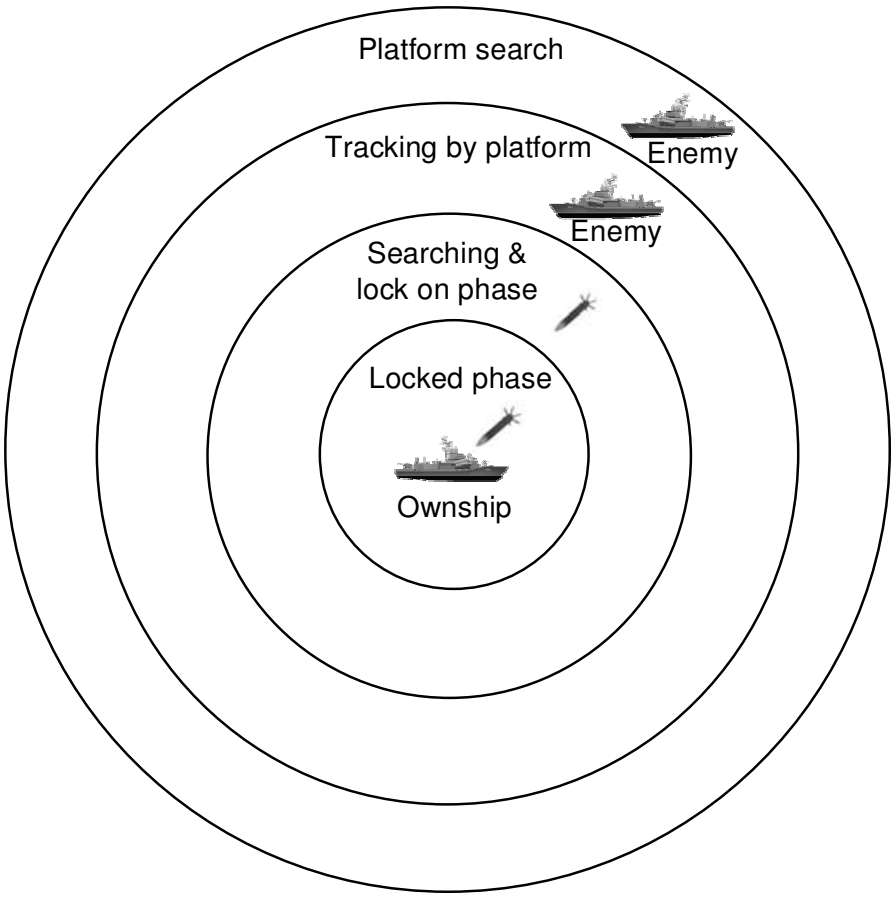



Figure 4.4: The four different attacking phases for a threat.

**Platform search** This area is where the enemy's platform (ship, plane, etc.) is searching for the ownship. Figure 4.5 describes such phase, where the radar of the enemy's platform is searching for the ownship. This radar has a beam circling around the enemy's platform with a certain range and speed. The ownship uses an Electronic Support Measures (ESM) radar to detect if an enemy's platform is searching. The ESM radar has a greater range than the enemy's radar since the energy it perceives has only traveled in one way. Indeed, the energy has to perform twice the distance to go back to the original platform. Therefore, the ship knows an enemy's is tracking to find another platform before the enemy may recognize its presence. Here, if the ownship has detected the enemy's platform, it can reduce its Radar Cross Section (RCS) to become more difficultly detectable. Then, the ship has to move away from the enemy's platform. Finally, the ownship has to consider the blind zones of the hardkill and softkill weapons while doing its manoeuvre.

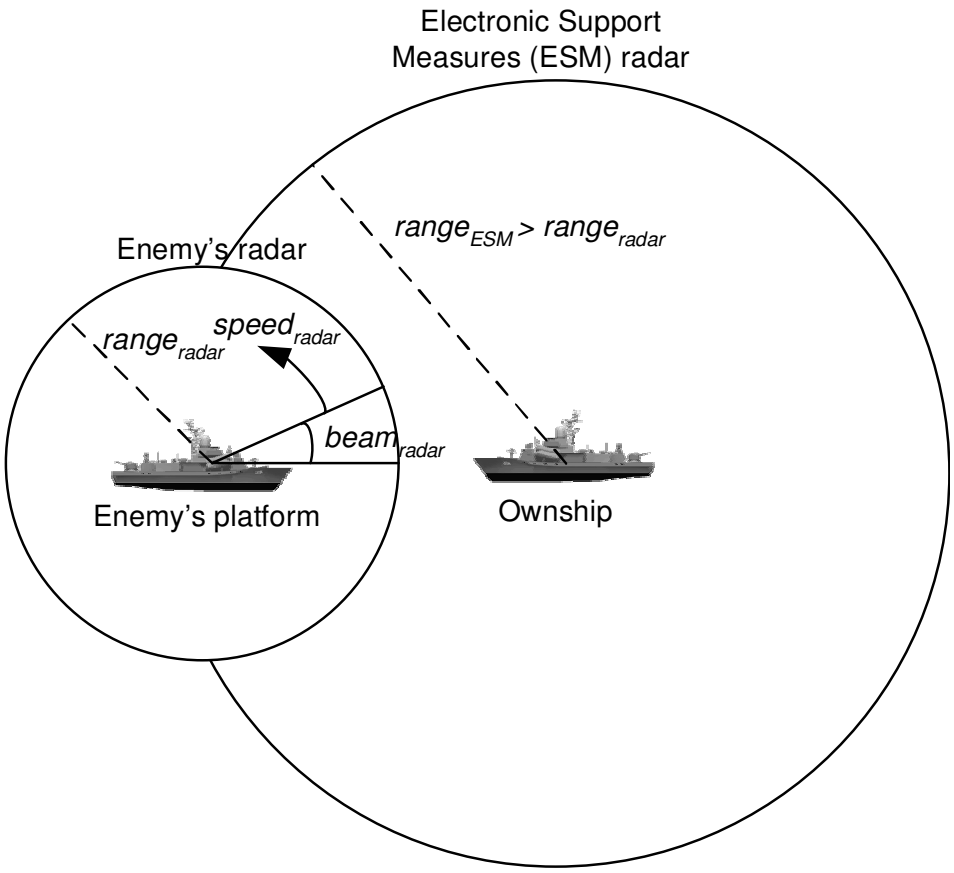



Figure 4.5: The *platform search* phase.

**Tracking by Platform** Before describing the *tracking by platform* phase, we introduce a jamming technique called *range-gate pull off*, as described in Figure 4.6. In short, the target uses a jamming to transmit the appropriate electromagnetic energy

back to the enemy's tracking Fire Control Radar (FCR) by increasing gradually the time $t$ the energy is transmitted. The enemy is, therefore, becoming increasingly desensitised to the real received energy, the enemy believes the ownship is moving away while it is not. Once this process has continued for a short period of time, the threat lost its fix on the ownship for another position.

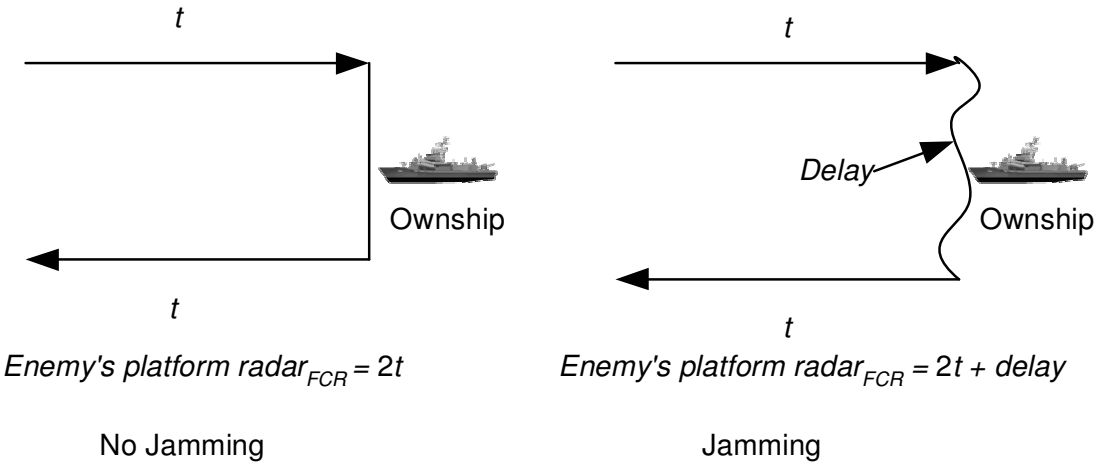



Figure 4.6: The *range-gate pull off* technique.

The tracking by platform & launch phase is described in Figure 4.7. This area reflects the zone where the FCR radar of the enemy's platform has located the ownship. The ownship knows it is being tracked because unlike in the tracking by platform phase where it received separate energy pulse, it receives a constant energy emission. To counter the enemy, the ownship can use the range-gate pull off technique. Then, it throws a chaff at the position the jamming deceived the enemy. Also, it has to consider the wind and other threats when throwing the chaff. Indeed, the ownship has to make it effective for the other incoming threats, too. Also, the wind has to help the ownship placing the chaff at its appropriate position. For example, it cannot throw a chaff with the wind in its front, because the chaff may return to the ownship. Then, it moves away from the gate of the FCR radar with a low RCS, and with a consideration of its blind zones incurred to other threats.

**Searching & Lock On** The enemy's platform has thrown an Anti Ship Missile (ASM) into the ownship direction. Firstly, the threat is ballistic as it is travelling in a straight line and its radar is not open, yet. The ASM knows its estimated distance from the ownship with the data transmitted by the FCR radar of the platform. At the time the distance from the ownship to the missile is the same, the missile opens its seeker at a random time. When the seeker is opened, it searches the ownship in a certain plan (see Figure 4.8). The ownship knows the threat is in this phase because the energy from the seeker is moving. To deceive the threat, the ship can diminish its

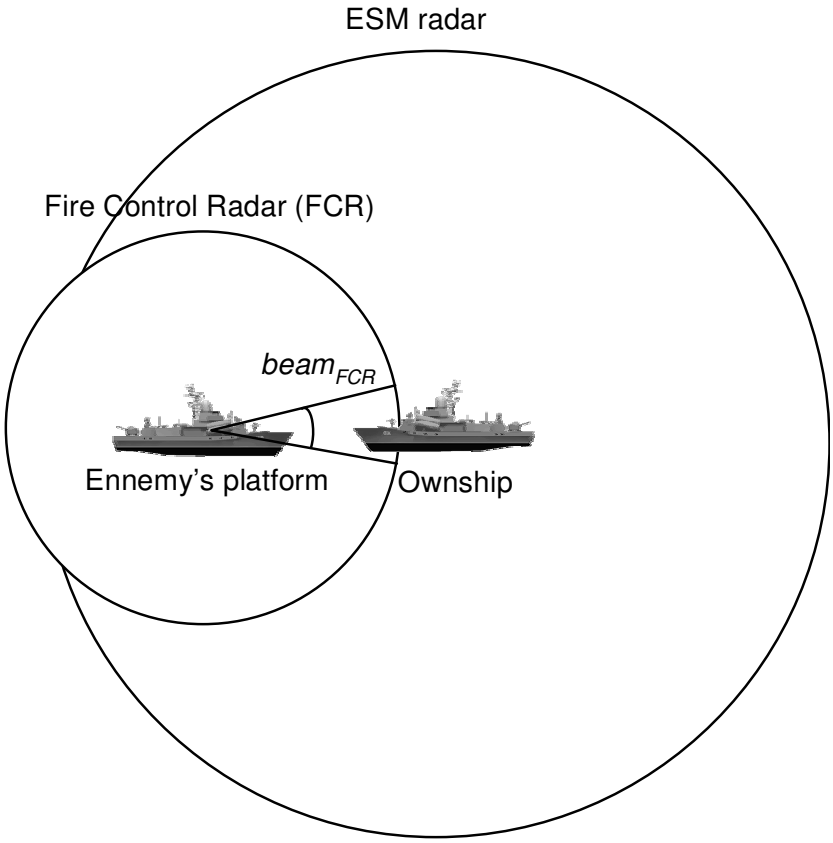



Figure 4.7: The *tracking by platform* phase.

RCS, considering its blind zones. Then, it has to throw a chaff to create an alternative target for the threat. In addition, it also has to consider other threats and the wind when it throws the chaff.

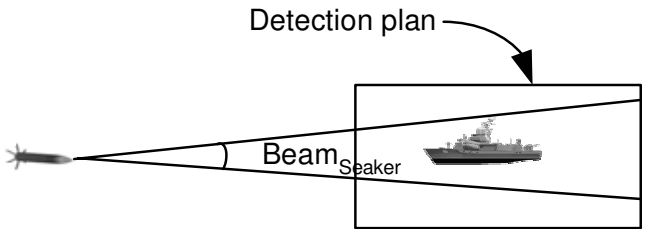



Figure 4.8: The *searching & lock on* phase.

**Locked** This phase is very similar to the tracking by platform one. The FCR of the threat has tracked the ownship. To counter the enemy, the ownship can use the range-gate pull off technique. Then, it throws a chaff at the position the jamming deceived the enemy. Furthermore, the ownship has to consider the wind and other threats when throwing the chaff. Indeed, it has to make it effective for the other incoming threats, too. Also, the wind has to help placing the chaff at its appropriate position. Then, the ship moves away from the gate of the FCR radar with a low RCS, and with a consideration of its blind zones incurred to other threats. Here, it has less time to quit the gate than in the tracking by platform phase because the threats travel at a higher speed. However, the gate produced by the radar of the threat is less than the one of the platform; as a consequence, the distance of the movement is less important.

All these phases have some coordination to be made when we are in a task group. For example, we don't want to throw a chaff near another ship and making the threat lock on it. This problem is very complicated since we have many planning, coordination, and prediction issues in it.

Resource allocation for naval engagements has three planning problems to solve to generate a plan: (1) Weapon-Target Assignment (WTA), (2) resource coordination, (3) movement. Indeed, a ship has firstly to assign weapons to some target. Secondly, these weapons should be coordinated altogether to avoid negative interactions and to profit from the positives ones. Finally, the movement of the ship should be considered. The next section introduces Weapon-Target Assignment which is a general model for resource allocation in military environments.

## 4.4 Weapon-Target Assignment

The *Weapon-Target Assignment* (WTA) problem is a fundamental problem arising in defense related applications of operations research (Ahuja et al., 2003). The problem consists of optimally assigning weapons to the enemy-targets so that the total expected survival value of the targets after all the engagements is minimized. There are two versions of the WTA problem: *static* (i.e. episodic) and *dynamic* (i.e. sequential). In the static version, all the inputs to the problem are fixed; that is, all targets are known, all weapons are known, and all weapons engage targets in a single stage. The dynamic version of the problem is a multi-stage problem where some weapons are engaged at the targets at a stage, the outcome of this engagement is assessed and strategy for the next stage is decided. This is called a "shoot-look-shoot-..." strategy since the defense is alternating between shooting its weapons and observing (looking) at the outcomes.

Efficient solutions of the WTA problem are of great interest to the military. The reason for this is that, in an engagement with the enemy, the problem must be solved in real-time. The enormous combinatorial complexity of the problem implies that, even with the supercomputers available today, optimal solutions cannot be obtained in real-time. One must therefore develop good heuristics for solving the problem (Hosein and Athans, 1990). To provide good heuristics one must have a thorough understanding of the properties of the problem and its solution. Some important properties of the dynamic WTA problem are:

- The computation time of any optimal algorithm for it grows exponentially with its size (NP-Completeness).
- It is sequential (the results of previous engagements are observed before making present assignments).
- It is nonlinear (the objective function is convex).
- It is stochastic (weapon-target engagements are modeled as stochastic events).
- It is large-scale (the number of weapons and targets is large, making enumeration techniques impractical)

Unfortunately, these properties of the problem rule out any hope of obtaining efficient optimal algorithms. Research on the WTA problem dates back to the 1950s and 1960s where the modeling issues for WTA problem were investigated (Manne (1958); Braford (1961); Day (1966)). Lloyd and Witsenhausen (1986) established the NP-completeness of the WTA problem. Exact algorithms have been proposed to solve the WTA problem for the following special cases: (i) when all the weapons are identical (DenBroader et al. (1958); Katter (1986)) or (ii) when the targets can receive at most one weapon

(Chang et al. (1987); Orlin (1987)). Some of the heuristics proposed to solve the WTA problem are based on nonlinear network flow (Castanon, 1987), neural networks (Wacholder, 1989), and genetic algorithms (Grant, 1993). Green et al. (1997) applied a goal programming-based approach to the WTA problem. Metler and Preston (1990) have studied a suite of algorithms for solving the WTA problem efficiently, which is critical for real-time applications of the WTA problem. Maltin (1970), Eckler and Burr (1972) and Murphey (1999) provide comprehensive reviews of the literature on the WTA problem. Research to date on the WTA. An approach either solves the WTA problem for special cases or develops heuristics for the WTA problem. Moreover, since no exact algorithm is available to solve the weapon target assignment problems, it is not known how accurate are the solutions obtained by these heuristic algorithms.

Ahuja et al. (2003) gave a mathematical formulation of the WTA problem. Let there be $n$ targets, numbered $1, 2, ..., n$ and $m$ weapon types, numbered $1, 2, ..., m$. Let $V_j$ denote the value of the target $j$, and $W_i$ denote the number of weapons of type $i$ available to be assigned to targets. Let $p_{ij}$ denote the probability of destroying target $j$ by a single weapon of type $i$. Hence $q_{ij} = 1 - p_{ij}$ denote the probability of survival of target $j$ if a single weapon of type $i$ is assigned to it. Observe that if we assign $x_{ij}$ number of weapons of type $i$ to target $j$, then the survival probability of target $j$ is given by $q_{ij}^{x_{ij}}$. A target may be assigned weapons of different types. The WTA problem is to determine the number of weapons $x_{ij}$ of type $i$ to be assigned to target $j$ to minimize the total expected survival value of all targets. This problem can be formulated as the following nonlinear integer programming problem:

$$\text{Minimise } \sum_{j=1}^{n} V_j \times \left( \prod_{i=1}^{m} q_{ij}^{x_{ij}} \right) \tag{4.1}$$

subject to

$$\sum_{j=1}^{n} x_{ij} \leqslant W_i, \text{ for all } i = 1, 2, ..., m,$$

$$x_{ij} \geqslant 0 \text{ and is an integer, for all } i = 1, 2, ..., m, \text{ and for all } j = 1, 2, ..., n.$$

In the above formulation, the expected survival value of the targets is minimized while ensuring that the total number of weapons used is no more than those available. This formulation presents a simplified version of the WTA problem. In more practical versions, adding additional constraints may be considered, such as (i) lower and/or upper bounds on the number of weapons of type $i$ assigned to a target $j$; (ii) lower and/or upper bounds on the total number of weapons assigned to target $j$; or (iii) a lower bound on the survival value of the target $j$. Notice that Equation 4.1 could be extended to consider positive and negative interactions between resources in the own platform if we are addressing WTA in the context of survival (Liang, 1995).

Ahuja et al. (2003) proposed several exact and heuristic algorithms to solve the WTA problem. Their branch and bound algorithms are the first implicit enumeration

algorithms that can solve moderately size instances of the WTA problem optimally. They also proposed a heuristic algorithm which improves iteratively an initial greedy solution and does not guarantee to obtain an optimal solution. This approach is similar to the tabu search algorithm used previously in our research (Plamondon (2003); Soucy (2003)).

### 4.4.1 Resource Allocation in a Platform

A ship contains two types of weapons. The *hardkill weapons* are used to counter-attack an Anti-Ship Missile (ASM) that comes toward the own platform. Their solution is to destroy the ASM in the air. There are three types of hardkill weapons : a long range Surface-to-Air Missile (SAM), a middle range gun, and a short range Close-In Weapons System (CIWS). The SAM and the gun are controlled by a Separate Tracking and Illuminating Radar (STIR), and the CIWS is controlled by the CIWS radar. On the other hand, the *softkill weapons* are used to confuse an ASM that comes toward the own platform. Their aim is to make the ASM fall in the water. There are two types of softkill weapons : The jamming and the chaff. The jamming aims to modify the waves of the radar that controls the threat that comes toward the own platform. The jamming tries to modify the destination of the ASM by controlling its own radar. The chaff is a cloud formed beside the ownship to make the threat's radar believes the cloud is the real ship. For instance, Figure 4.9 represents the interactions between the STIR and the chaff. A kill assessment operation follows each engagement to measure its outcome (Plamondon (2003); Frini et al. (2004)). The outcome is assumed to be binary, i.e., a total success or a total failure. Thus, after each engagement, threat can be "killed" or "not killed". The probability of the state "killed" is calculated by

$$P_K = 1 - P_{NK} = 1 - \prod_{w \in W} (1 - P_K^w) \tag{4.2}$$

The resources of the own platform can be modeled as a WTA problem. In particular, the own platform has $m$ weapon types: SAM, Gun, CIWS, Jamming and Chaff. Each weapon of type $i$ is constrained on the amount ($W_i$) that may be used to counter the incoming $n$ missile target types. For example, if the own platform possesses two STIRs, we can use only two SAMs/Guns at a time. Also, each weapon of type $i$ has a probability $p_{ij}$ to counter each target $j$. Furthermore, each target $j$ has a value ($V_j$) characterized by its range, azimuth and dangerousness. As we can see, all the required variables for the WTA problem are present in the resource allocation of the own platform.

## 4.5 Resource Coordination

It is necessary to coordinate effectively the weapons, sensors and manoeuvres for a ship. Liang (1995) and Liang and Liem (1992) discuss the problematic created by the different positive and negative interactions between the resources of a ship. These interactions are separated into three types: positive, neutral and negative. For example, when using a STIR, we emit energy that the threat can sense, thus making the chaff efficiency much lower. Indeed, using a conjunction of a STIR and a chaff in the same area forms a negative interaction. It is necessary to manage very well such resources to make positive interactions effective and avoid negative interactions. Elsewhere, we investigated many coordination strategies to manage efficiently these interactions. The results suggested that a central coordinator technique provides a coordinated plan in a minimal time (Plamondon, 2003).

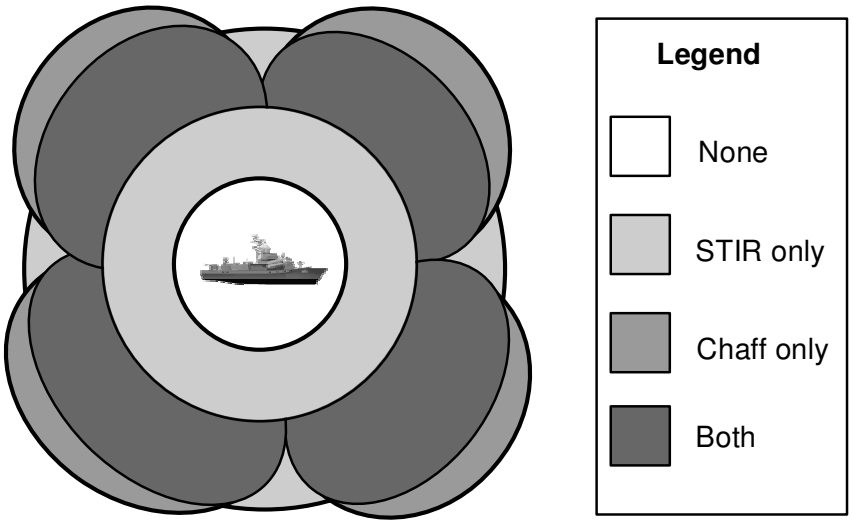



Figure 4.9: Interactions between the STIR and the chaff in a ship (from Liang (1995)).

Furthermore, Visser (2001) indicated that current anti-ship missile defence is limited to preplanned actions, voice commands during engagement, and relatively simple computer-based advice (if all present). However, none of these command procedures is considered to be fast and accurate enough once a multiple anti-ship attack has begun. Visser has participated in a research program in order to develop a technology demonstrator in which rules for coordinated hardkill and softkill weapon deployment could be evaluated. The demonstrator of such researches consisted of a hardkill scheduler, a softkill scheduler, a hardkill/softkill coordinator and a simulation environment to test and evaluate these subsystems. The rules concerning the coordination between the hardkill and softkill weapon systems have been incorporated in a third module (the hardkill/softkill coordinator). The softkill deployment rules were optimized in the

sense that negative interactions between different softkill measures are avoided and that positive synergy between different softkill measures is achieved as much as possible. A scheduler has also been considered in order to take into account the effects on other engaged threats, the availability of softkill assets, the wind and the geometry of a ship.

In the meanwhile, Oard et al. (1994) have considered the problem of integrated and coordinated employment of cruise missile defenses and Chaffs on a single ship. A mathematical model of their performance has been developed. An optimal scheduling problem was posed using this model, and a dynamic programming solution was developed. Because value iteration dynamic programming required working backward at time, they considered every state combination of state variables at every time. Although they are able to find optimal solutions small problems using dynamic programming, they claimed that solving problems based on larger models requires more computer resources than can practically be provided. For this reason, they were motivated to search for faster techniques. Both heuristic techniques and alternatives uses of the dynamic programming equations bear further investigations. They also stated that the value iteration dynamic programming solution they have developed is useful, however, since it can be used to gain insight into the design of such algorithms and to evaluate the performance or those algorithms in a restricted domain.

Brown et al. (2001) are, on the other hand, investigating innovative real-time decision support concepts to aid the commander to coordinate many platforms together. Precisely, their research scope have addressed the following areas:

- display of battle information to the commander that can be easily assimilated, reasoned and actioned upon;
- rapid evaluation of threats at the force level;
- generating and analyzing complex force response options involving combinations of both hard and soft kill weapons (hardkill/softkill);
- providing the command with a set of prioritised response options (recommendations);
- continuous engagement monitoring, assessment and re-allocation across the force;
- fully automatic capability for complex and/or time critical situations;
- distributed implementation for improved speed and robustness;
- real-time implementation;
- hardkill/softkill co-ordination techniques;

- information exchange requirements.

The next section describe a brief literature review on own platform movement.

## 4.6 Movement

Very few researchers have addressed the ship manoeuvre. Except, Hong et al. (2003) who proposed a collision avoidance ship manoeuvre method. Ghose (2003), for his part, proposed a missile engagement navigation system. Both these works are not related on the specific problem we are working on: Minimizing our blind zones and minimizing our surface exposed to incoming threats. The only deep work on these problems were performed by Plamondon (2003), Morissette et al. (2004) and Plamondon et al. (2003). To this end, a first positioning strategy by Plamondon (2003) separates the environment in different sectors of effectiveness for the own platform. Then, a position is made to maximize the effectiveness for the own platform to counter the incoming threats by changing the respective sector of the threats using a naïve Bayes classifier heuristic. Another method from Morissette et al. (2004), which also uses the sector of effectiveness, has been using an heuristic which evaluates the effectiveness of a position according to the threats found in the environment. Then, an algorithm treats all the possible rotations and suggests the best regarding a given situation. In this context, a first version of a Radar Cross Section (RCS) reduction movement was proposed (Plamondon et al., 2003).

We now introduce a simple resource allocation problem and the problem instance we used in this thesis, which retains all the characteristics of the problem described in this chapter.

## 4.7 Domain of Experiment for the Thesis

The domain of the experiments in this thesis is as following. The domain is a naval platform which must counter incoming missiles (i.e. tasks) by using its resources (i.e. weapons, movements). For the experiments, 100 randomly resource allocation problems were generated for each approach, and possible number of tasks.

In our problem, $|S_{ta}| = 4$, thus each task can be in four distinct states. There are two types of states; firstly, states where actions modify the transition probabilities; and then, there are goal states. The state transitions are all stochastic because when a missile is in a given state, it may always transit in many possible states. In particular, each resource type has a probability to counter a missile between 45% and 65% depending on the state of the task. When a missile is not countered, it transits to another state,

which may be preferred or not to the current state, where the most preferred state for a task is when it is countered. The effectiveness of each resource is modified randomly by ±15% at the start of a scenario.

There are also local and global resource constraints on the amount that may be used. For the local constraints, at most 1 resource of each type can be allocated to execute tasks in a specific state. This constraint is also present on a real naval platform because of sensor and launcher constraints and engagement policies. Furthermore, for consumable resources, the total amount of available consumable resource is between 1 and 2 for each type. The global constraint is generated randomly at the start of a scenario for each consumable resource type. Temporal constraints are not considered in this problem for simplicity. Still, as explained in Section 6.1.2, $|A|$ is combinatorial with the number of resource types.

Chapter 7 tackles a much more complex problem, which takes into account all main aspect of scheduling, including temporal constraints. To test some algorithms, some modifications of these settings have been made and they are detailed.

### 4.7.1 Discussion

As explained in Section 4.3.2, the naval above-water warfare resource management problem for a ship falls into the *partially observable*, *stochastic*, *sequential*, *dynamic*, *discrete*, and *multiagent* case. Table 4.1 summarizes the main characteristics of the artificial intelligence and operation research/decision sciences planning approaches.

| Ship environment / Approaches | Stochastic | Observation | Optimality |
|---|---|---|---|
| MDPs | Yes | Yes | Yes |
| Classical planning | No | No | No |
| Probabilistic planning | Yes | Yes | No |
| Contingency planning | Yes | Yes | No |

Table 4.1: Identification of the planning approach for a ship.

An MDP approach permits to compare its solution (policy) to the optimal solution and provides a convenient representation of the state space. An MDP representation is very general and modeling the problem with WTA of design-to-time scheduling results only in a specialization of the model.

However, one may claim, that POMDPs are more suitable, but the complexity of this approach and the fact that a MDP is only a specialization of POMDPs keeps the MDP framework valid. Indeed, once our problem is well solved using the MDP approach, extending it to a partially observable environment does not modify the validity

of our work. In the previous section, we discussed that our motivated problem requires resource coordination and movement. Using MDPs, the resource coordination problem can be solved by modifying the transition probabilities of using a resource on a specific threat, if another conflicting resource is used simultaneously. Furthermore, the movement is simply a possible action to execute by the MDP planner.

### 4.7.2 Resource Allocation as a MDPs

In our domain of experiment, presented in Section 4.7, the transition function and the reward function are both known. A Markov Decision Process (MDP) framework is used to model our stochastic resource allocation problem. MDPs have been widely adopted by researchers today to model a stochastic process. This is due to the fact that MDPs provide a well-studied and simple, yet very expressive model of the world. An MDP in the context of a resource allocation problem with limited resources is defined as a tuple $\langle Res, Ta, S, A, P, W, R, \rangle$, where:

- $res \in Res$ is a finite set of resource types available for a planning process. Each resource type may have a local resource constraint $L_{res}$ on the number that may be used in a single step, and a global resource constraint $G_{res}$ on the number that may be used in total. The global constraint only applies for consumable resource types ($Res_c$) and the local constraints always apply to consumable and non-consumable resource types.

- $Ta$ is a finite set of tasks with $ta \in Ta$ to be accomplished.

- $S$ is a finite set of states with $s \in S$. A state $s$ is a tuple $\langle Ta, Res_c \rangle$, which is the characteristic of each unaccomplished task $ta \in Ta$ in the environment, and the available consumable resources. $s_{ta}$ is the specific state of task $ta$. Also, $S$ contains a non empty set $s_g \subseteq S$ of goal states. A goal state is a sink state where an agent stays forever.

- $A$ is a finite set of actions (or assignments). The actions $a \in A(s)$ applicable in a state are the combination of all resource assignments that may be executed, according to the state $s$. In particular, $a$ is simply an allocation of resources to the current tasks, and $a_{ta}$ is the resource allocation to task $ta$. The possible actions are limited by $L_{res}$ and $G_{res}$.

- Transition probabilities $P_a(s'|s)$ for $s \in S$ and $a \in A(s)$.

- $W = [w_{ta}]$ is the relative weight (criticality) of each task.

- State rewards $R = [r_s] : \sum_{ta \in Ta} r_{s_{ta}} \leftarrow \Re_{s_{ta}} \times w_{ta}$. The relative reward of the state of a task $r_{s_{ta}}$ is the product of a real number $\Re_{s_{ta}}$ by the weight factor $w_{ta}$. For

our problem, a reward of $1 \times w_{ta}$ is given when the state of a task ($s_{ta}$) is in an achieved state, and 0 in all other cases.

- A discount (preference) factor $\gamma$, which is a real number between 0 and 1.

A solution of an MDP is a policy $\pi$ mapping states $s$ into actions $a \in A(s)$. In particular, $\pi_{ta}(s)$ is the action (i.e. resources to allocate) that should be executed on task $ta$, considering the global state $s$. In this case, an optimal policy is one that maximizes the expected total reward for accomplishing all tasks. The optimal value of a state, $V(s)$, is given by Equation 3.2 where the remaining consumable resources in state $s'$ are $Res_c \setminus res(a)$, where $res(a)$ are the consumable resources used by action $a$. Indeed, since an action $a$ is a resource assignment, $Res_c \setminus res(a)$ is the new set of available resources after the execution of action $a$. The policy is subjected to the local resource constraints $res(\pi(s)) \leqslant L_{res} \forall\ s \in S$ , and $\forall\ res \in Res$. The global constraint is defined according to all system trajectories $tra \in TRA$. A system trajectory $tra$ is a possible sequence of state-action pairs, until a goal state is reached under the optimal policy $\pi$. For example, state $s$ is entered, which may transit to $s'$ or to $s''$, according to action $a$. The two possible system trajectories are $\langle (s, a), (s') \rangle$ and $\langle (s, a), (s'') \rangle$. The global resource constraint is $res(tra) \leqslant G_{res} \forall\ tra \in TRA$ ,and $\forall\ res \in Res_c$ where $res(tra)$ is a function which returns the resources used by trajectory $tra$. Since the available consumable resources are represented in the state space, this condition is verified by itself. In other words, the model is Markovian as the history has not to be considered in the state space. Furthermore, the time is not considered in the model description, but it may also include a time horizon by using a finite horizon MDP.

## 4.8 Conclusion

This chapter detailed the specific problematic and and the domain of experiment for the thesis. Our problematic contains three main aspects: Weapon-target assignment, resource coordination and ship movement. We have chosen to model this resource allocation problem with Markov Decision Processes (MDPs) because it is very suitable.

The next chapter introduces heuristic search which is used in Sections 5.2, 6.1 and Chapter 7 to prune the state and action spaces of a resource allocation problem.

# Chapter 5

# Heuristic Search Approaches

## 5.1 Overview

A common way of addressing the large stochastic problem of resource allocation is by using Markov Decision Processes (MDPs), and in particular real-time search where many algorithms have been developed recently. For instance Real-Time Dynamic Programming (RTDP) (Barto et al., 1995), LRTDP (Bonet and Geffner, 2003b), HDP (Bonet and Geffner, 2003a), and LAO$^\star$ (Hansen and Feng, 2001) are all state-of-the-art heuristic search approaches in a stochastic environment.

An interesting approach, by its anytime quality, is RTDP introduced by Barto et al. (1995) which updates states in trajectories from an initial state $s_0$ to a goal state $s_g$. RTDP is much more effective if the generated trajectories are efficient. To achieve this, Bounded RTDP (BRTDP) (McMahan et al., 2005), Focused RTDP (FRTDP) (Smith and Simmons, 2006), and modified value iteration (Singh and Cohn, 1998) are approaches for solving a stochastic problem using a RTDP type heuristic search with upper and lower bounds on the value of states.

An optimal policy can be found using an off-line dynamic programming algorithm such as policy iteration or value iteration. But a disadvantage of dynamic programming is that it evaluates the entire state space. In effect, it finds a policy for every possible starting state. By contrast, heuristic search algorithms solve a problem for a particular starting state and use an admissible heuristic to focus the search, and remove from consideration regions of the state space that cannot be reached from the start state by an optimal solution. For problems with large state spaces, heuristic search has an advantage over dynamic programming because it can find an optimal solution for a start state without evaluating the entire state space.

This advantage is well-known for problems that can be solved by A$^*$ (P. Hart, 1968). In fact, an important theorem about the behavior of A$^*$ is that (under certain condi-

tions) it evaluates the minimal number of states among all algorithms that find an optimal solution (Dechter and Pearl, 1985). The A* algorithm lies the foundation of all heuristic search algorithm presented in this chapter.

### 5.1.1 RTDP

RTDP (Barto et al., 1995) has emerged from Learning Real-Time A* (LRTA*) (Korf, 1990), which in turn, has emerged from A* (P. Hart, 1968). We should note that A* relies on a heuristic function $h(s)$ that estimates the cost from state $s$ to the goal. Not learning the $h(s)$ function in this algorithm may cause two problems: it may return non-optimal solutions, or it may *loop* and return no solution at all. Korf showed that updating the utilities each time a state is visited can solve both these problems. Indeed, LRTA* guarantees that the search will not be trapped into a loop as long as there is a path from every state to the goal. Second, Korf proved that *successive trials* of the algorithm, each beginning with the value function resulting from the previous trial, eventually deliver an optimal path to the goal. This is provided that the heuristic $h$ used to initialize the value function $V$ is *admissible*[1]. The rate at which LRTA* converges to the optimal solution depends on the size of the state space and the quality of the heuristic function $h(s)$. A better heuristic yields a more focussed search, a high ratio of updates on the relevant states, and a faster convergence.

**Algorithm 5.1** The RTDP algorithm (Barto et al., 1995).

1: **Function** RTDP($S$)
2: **returns** a value function $V$
3: **repeat**
4: $s \leftarrow s_0$
5: **repeat**
6: $V(s) \leftarrow R(s) + \max_{a \in A(s)} \gamma \sum_{s' \in S} P_a(s'|s)V(s')$
{where $V(s') = h(s')$ when $s'$ is not yet visited, and $s'$ has $Res_c \setminus res(a)$ remaining consumable resources}
7: $Res_c \leftarrow s.Res_c \setminus res(\pi(s))$
8: $s \leftarrow s$.PICK-NEXT-STATE($Res_c$)
9: **until** $s$ is a goal
10: **until** No more computing time is allowed
11: **return** $V$

RTDP (Barto et al., 1995) (Algorithm 5.1) is a probabilistic version of LRTA* and

[1]When we want to minimize the cost, an admissible heuristic is one that affects a greater (lesser when we maximize the expected reward) value for all state than the "real" one. This way, all relevant states are assured of being explored.

has the same properties. A good advantage of RTDP, just like all MDP algorithms, is that it is an anytime algorithm. This algorithm can also be viewed as a greedy version of the dynamic programming algorithms for solving MDPs. Each RTDP trial, or trajectories (Line 6 to 11 of the Algorithm 5.1), is the result of simulating the policy $\pi$, through the PICK-NEXT-STATE($Res_c$) function, while updating the values $V(s)$ using a Bellman backup (Equation 3.2) over the states $s$ that are visited. $h(s')$ is a heuristic which defines an initial value for state $s'$. This heuristic has to be admissible — The value given by the heuristic has to overestimate (or underestimate) the optimal value when the objective function is maximized (or minimized). For example, an admissible heuristic for a stochastic shortest path problem is the solution of a deterministic shortest path problem. Indeed, since the problem is stochastic, the optimal value is lower than this one given by the deterministic version. Furthermore, RTDP interleaves planning and execution, and in this sense it is an on-line algorithm. Unfortunately, the backups are made infinitely often in the limit for RTDP, and we can never know when the algorithm has converged. For this reason, the algorithm stops to perform trajectories when a time deadline is reached (Line 10 of Algorithm 5.1.

### 5.1.2 LRTDP

Bonet and Geffner (2003b) proposed LRTDP (Algorithm 5.2) as an improvement to RTDP (Barto et al., 1995). LRTDP is a simple dynamic programming algorithm that involves, like RTDP, a sequence of trial runs. Each trajectories starts in the initial state $s_0$ and ends in a goal, i.e. a *solved* state in this approach.

It has been proven that LRTDP, given an admissible initial heuristic on the value of states cannot be trapped in loops, and eventually yields optimal values. The convergence is accomplished by means of a labeling procedure called CHECK-SOLVED($s$) in Line 14 of the LRTDP function. The CHECK-SOLVED function is presented in Algorithm 5.3. This procedure tries to label as solved each traversed state in the current trajectory. A state $s$ is labeled as solved iff $s.solved = true$. When the initial state is labelled as solved, the algorithm has converged. Note that the depth first search stores all states visited in a closed list (Line 11) to avoid loops and duplicate work. Thus the time and space complexity of CHECK-SOLVED($s$) is $O(S)$ in the worst case, while a tighter bound is given by $O(S_V(s))$ where $S_V(s) \subseteq S$ refers to the states that are reachable from a state $s$ with the initial heuristic policy. This distinction is important as the reachable states from $s$ may be much smaller than the complete state space in certain cases depending on the domain and the quality of the initial value (heuristic) function.

**Algorithm 5.2** The LRTDP algorithm (Bonet and Geffner, 2003b).

1: **Function** LRTDP($S$)
2: **returns** a value function $V$
3: **repeat**
4: $s \leftarrow s_0$
5: $visited \leftarrow null$
6: **repeat**
7: $visited.push(s)$
8: $V(s) \leftarrow R(s) + \max_{a \in A(s)} \gamma \sum_{s' \in S} P_a(s'|s)V(s')$
{where $V(s') = h(s')$ when $s'$ is not yet visited, and $s'$ has $Res_c \setminus res(a)$ remaining consumable resources}
9: $Res_c \leftarrow s.Res_c \setminus res(\pi(s))$
10: $s \leftarrow s$.PICK-NEXT-STATE($Res_c$)
11: **until** $s$ is a goal
12: **while** $visited \neq null$ **do**
13: $s \leftarrow visited.pop()$
14: **if** $\neg$ CHECK-SOLVED($s$) **then**
15: break
16: **end if**
17: **end while**
18: **until** $s_0$ is *solved*
19: **return** $V$

### 5.1.3 RTDP with Two Bounds

RTDP is much more effective if the generated trajectories are efficient. To achieve this, Bounded RTDP (BRTDP) (McMahan et al., 2005), Focused RTDP (FRTDP) (Smith and Simmons, 2006), and modified value iteration (Singh and Cohn, 1998) are approaches for solving a stochastic problem using a RTDP type heuristic search with upper and lower bounds on the value of states.

We can supply RTDP by two bounds through two distinct initial heuristics for unvisited states $s \in S$: $h_L(s)$ and $h_U(s)$. On the one hand, $h_L(s)$ defines a lower bound on the value of $s$ such that the optimal value of $s$ is higher than $h_L(s)$. On the other hand, $h_U(s)$ defines an upper bound on the value of $s$ such that the optimal value of $s$ is lower of equal than $h_U(s)$. Also, $L(s)$ is the lower bound value of state $s$, while $U(s)$ is the upper bound value of state $s$. Similarly, $Q_L(a, s)$ is the Q-value of the lower bound of action $a$ in state $s$, while $Q_U(a, s)$ is the Q-value of the upper bound of action $a$ in state $s$.

**Algorithm 5.3** The CHECK-SOLVED algorithm for LRTDP (Bonet and Geffner, 2003b).

1: **Function** CHECK-SOLVED($s$)
2: **returns** a boolean $rv$
3: $rv \leftarrow true$
4: $open \leftarrow$emptyStack
5: $closed \leftarrow$emptyStack
6: **if** $\neg s$.SOLVED **then**
7: $open$.PUSH($s$)
8: **end if**
9: **while** $open \neq$ emptyStack **do**
10: $s \leftarrow open$.POP()
11: $closed$.PUSH($s$)
{check residual}
12: **if** $\delta(s) > \epsilon$ **then**
13: $rv \leftarrow false$
14: **continue**
15: **end if**
16: **for all** $s'$ such that $P_{\pi(s)}(s'|s) > 0$ **do**
17: **if** $\neg s$.SOLVED $\wedge \neg$ IN($s', open \cup closed$) **then**
18: $open$.PUSH($s$)
19: **end if**
20: **end for**
21: **end while**
22: **if** $rv = true$ **then**
23: {label relevant states}
24: **for all** $s' \in closed$ **do**
25: $s'$.SOLVED $= true$
26: **end for**
27: **else**
28: **while** $closed \neq$ emptyStack **do**
29: $s \leftarrow closed$.POP()
30: $V(s) \leftarrow R(s) + \max_{a \in A(s)} \gamma \sum_{s' \in S} P_a(s'|s) V(s')$
31: **end while**
32: **end if**
33: **return** $rv$

Also, the CHECK-SOLVED($s$, $\epsilon$) procedure can be omitted from RTDP with two bounds approaches because the bounds can provide the labeling of a state as solved.

Computing these two bounds is made simultaneously as the state transitions are the same for both bounds. Only the values of the state transitions change. Thus, having to compute two Q-values instead of one does not augment the complexity of the bounded approaches. In fact, Smith and Simmons (2006) state that the additional time to compute a Bellman backup for two bounds, instead of one, is no more than 10%.

As discussed by Singh and Cohn (1998), RTDP with two bounds has a number of desirable anytime characteristics: if an action has to be picked in state $s$ before the algorithm has converged (while multiple competitive actions remains), the action with the highest lower bound is picked. Since the upper bound for state $s$ is known, it may be estimated how far the lower bound is from the optimal. If the difference between the lower and upper bound is too high, one can use another greedy algorithm with a fast and near optimal solution, if it is possible. The following sections present three types of RTDP with two bounds type of algorithms: BRTDP, FRTDP, and modified value iteration.

### BRTDP

The pseudocode for Bounded RTDP (BRTDP) (McMahan et al., 2005) is given in Algorithm 5.4. BRTDP has many differences from RTDP: The first is when a policy is requested for BRTDP(before or after convergence), it is returned based on the lower bound $L$. The second difference is that $L$ helps guide exploration in simulation, as computed in Lines 11 to 16 of the algorithm. In particular, when trajectories are sampled in simulation, the outcome distribution is biased to prefer transitions to states with a large gap $(U(y) - L(y))$. Furthermore, BRTDP maintains a list of states on the current trajectory, and when the trajectory terminates, it does backups in reverse order along the stored trajectory (Lines 18 to 22 of the BRTDP function). Finally, like LRTDP, simulated trajectories terminate when they reach a state that has a "well-known" value, rather than when they reach the goal. McMahan et al. (2005) proved the convergence of BRTDP.

### FRTDP

Focused RTDP (Algorithm 5.5) is also an RTDP based algorithm proposed by Smith and Simmons (2006). As in RTDP, FRTDP's execution consists in trials that begin in a given initial state $s_0$ and then explore reachable states of the state space, selecting actions according to an upper bound. Once a final state is reached, it performs Bellman updates on the way back to $s_0$.

FRTDP uses a priority value for selecting actions outcomes and detecting trial

**Algorithm 5.4** The BRTDP algorithm (McMahan et al., 2005).

1: **Function** BRTDP($S$)
2: **returns** a value function $V$
3: **while** $U(s_0) - L(s_0) > \epsilon$ **do**
4: $x \leftarrow s_0$
5: $traj \leftarrow$ EMPTYSTACK
6: **while** true **do**
7: $traj$.PUSH($x$)
8: $U(x) \leftarrow R(x) + \gamma \sum_{x' \in S} P_a(x'|x)U(x')$
{where $U(x) \leftarrow h_U(x)$ when $x$ is not yet visited and $x'$ has $Res_c \setminus res(a)$ remaining consumable resources}
9: $a \leftarrow \arg\max_{a \in A(s)} Q_L(x, a)$
10: $L(x) \leftarrow R(x) + \gamma \sum_{x' \in S} P_a(x'|x)L(x')$
{where $L(x) \leftarrow h_L(x)$ when $x$ is not yet visited and $x'$ has $Res_c \setminus res(a)$ remaining consumable resources}
11: $\forall y, b(y) \leftarrow P_a(y|x)(U(y) - L(y))$
12: $B \leftarrow \sum_{y \in S} b(y)$
13: **if** $B < (U(s) - L(s))/\tau$ **then**
14: break
15: **end if**
16: $x \leftarrow$ sample from distribution $b(y)/B$
17: **end while**
18: **while** $traj \neq null$ **do**
19: $x \leftarrow traj$.POP()
20: $U(x) \leftarrow R(x) + \gamma \sum_{x' \in S} P_a(x'|x)U(x')$
21: $L(x) \leftarrow R(x) + \gamma \sum_{x' \in S} P_a(x'|x)L(x')$
22: **end while**
23: **end while**
24: **return** $V$

termination (Lines 7 to 9 of the BACKUP function). The lower bound is used to establish the policy by contributing in the priority calculation of states to expand on the fringe of the search tree. Furthermore, in this algorithm, trial termination detection has been modified and improved from RTDP by adding an adaptive maximum depth $D$ in the search tree in order to avoid over-committing to long trials early on (Line 4 of the TRIAL function). In fact, the maximum depth $D$ is updated by $k_D \times D$ in Line 7 of the FRTDP function each time the trial is not useful enough. This usefulness is

represented by $\delta W$ where $\delta$ measures how much the update changed the upper bound value of $s$ and $W$ the expected amount of time the current policy spends in $s$, adding up all possible paths from $s_0$ to $s$.

**Algorithm 5.5** The FRTDP algorithm (Smith and Simmons, 2006).

**Function** FRTDP($s_0$)
$D \leftarrow D_0$
**while** $U(s_0) - L(s_0) > \varepsilon$ **do**
  $(q_p, n_p, q_c, n_c) \leftarrow (0, 0, 0, 0)$
  TRIAL($s_0$,$W = 1$,$d = 0$)
  **if** $(q_c/n_c) \geqslant (q_p/n_p)$ **then**
    $D \leftarrow k_D D$
  **end if**
**end while**

**Function** INIT($s$) {implicitly called the first time each state $s$ is touched}
$(L(s), U(s)) \leftarrow (h_L, h_U)$
$s.prio \leftarrow \Delta(s)$

**Function** TRIAL($s$, $W$, $d$)
$(a^*, s^*, \delta) \leftarrow$ BACKUP($s$)
TRIAL-UPDATE($\delta W, d$)
**if** $\Delta(s) \leqslant 0$ **or** $d \geqslant D$ **then**
  **return**
**end if**
TRIAL($s^*$,$\gamma P_{a^*}(s^*|s)W$,$d + 1$)
BACKUP($s$)

**Function** TRIAL-UPDATE($q, d$)
**if** $d > D/k_D$ **then**
  $(q_c, n_c) \leftarrow (q_c + q, n_c + 1)$
**else**
  $(q_p, n_p) \leftarrow (q_p + q, n_p + 1)$
**end if**

**Function** BACKUP($s$)
$L(s) \leftarrow \max_{a \in A(s)} \mathrm{QL}(a, s)$
$u \leftarrow \max_{a \in A(s)} \mathrm{QU}(a, s)$
$a^* \leftarrow \arg\max_a \mathrm{QU}(a, s)$
$\delta \leftarrow |U(s) - u|$
$U(s) \leftarrow u$
$p \leftarrow \max_{s' \in S} \gamma P_{a^*}(s'|s) s'.prio$
$s^* \leftarrow \arg\max_{s' \in S} \gamma P_{a^*}(s'|s) s'.prio$
$s.prio \leftarrow \min(\Delta(s), p)$
**return** $(a^*, s^*, \delta)$

**Function** $\Delta(s)$
**return** $|U(s) - L(s)| - \varepsilon/2$

**Function** QL($a, s$)
**return** $R(s) + \gamma \sum_{s' \in S} \gamma P_a(s'|s) L(s')$ {where $L(s) \leftarrow h_L(s)$ when $s$ is not yet visited and $s'$ has $Res_c \setminus res(a)$ remaining consumable resources}

**Function** QU($a, s$)
**return** $R(s) + \gamma \sum_{s' \in S} \gamma P_a(s'|s) U(s')$ {where $U(s) \leftarrow h_U(s)$ when $s$ is not yet visited and $s'$ has $Res_c \setminus res(a)$ remaining consumable resources}

### Modified Value Iteration Algorithm

The modified value iteration algorithm proposed by Singh and Cohn (1998) is presented in Algorithm 5.6. An interesting characteristic of this algorithm is to use the bounds for

significantly reducing the action space $A$. Indeed, in Lines 5 and 6 of the BOUNDED-BACKUP function, if $Q_U(a,s) \leqslant L(s)$ then action $a$ may be pruned from the action space of $s$. Singh and Cohn (1998) proved that an algorithm that uses admissible lower and upper bounds to prune the action space is assured of converging to an optimal solution.

**Algorithm 5.6** The modified value iteration algorithm (Singh and Cohn, 1998).

1: **Function** MODIFIED-VI($S$)
2: **returns** a value function $V$
3: **repeat**
4: $s \leftarrow s_0$
5: $visited \leftarrow null$
6: **repeat**
7: $visited.push(s)$
8: BOUNDED-BACKUP($s$)
9: $Res_c \leftarrow s.Res_c \setminus \{\pi(s)\}$
10: $s \leftarrow s$.PICK-NEXT-STATE($Res_c$)
{Select a possible transiting state $s'$ with the highest $U(s') - L(s')$}
11: **until** $s$ is a goal
12: **until** $|A(s)| = 1$(or the Bellman error of $s$ is bounded by $\epsilon$) $\forall\ s \in S$ reachable from the initial state $s_0$
13: **return** $V$

**Algorithm 5.7** The bounded Bellman backup for modified value iteration algorithm.

1: **Function** BOUNDED-BACKUP($s$)
2: **for all** $a \in A(s)$ **do**
3: $Q_U(a,s) \leftarrow R(s) + \gamma \sum_{s' \in S} P_a(s'|s) U(s')$
4: $Q_L(a,s) \leftarrow R(s) + \gamma \sum_{s' \in S} P_a(s'|s) L(s')$
{where $L(s') \leftarrow h_L(s')$ and $U(s') \leftarrow h_U(s')$ when $s'$ is not yet visited and $s'$ has $Res_c \setminus res(a)$ remaining consumable resources}
5: **if** $Q_U(a,s) \leqslant L(s)$ **then**
6: $A(s) \leftarrow A(s) \setminus a$
7: **end if**
8: **end for**
9: $L(s) \leftarrow \max_{a \in A(s)} Q_L(a,s)$
10: $U(s) \leftarrow \max_{a \in A(s)} Q_U(a,s)$
11: $\pi(s) \leftarrow \arg\max_{a \in A(s)} Q_L(a,s)$

**Comparison of FRTDP, BRTDP and Modified Value Iteration**

Both FRTDP and BRTDP propose an efficient trajectory of state updates to further speed up the convergence, when given upper and lower bounds. BRTDP selects a state $s'$ from a random distribution $b(s')$, such as:

$$\sum_{s' \in S} b(s') = P_{\pi(L(s))}(s'|s)(U(s') - L(s'))$$

. Similarly, FRTDP selects the next state according to a state priority $p(s)$. $p(s) = U(s) - L(s)$ for fringe (not expanded) node, and

$$p(s) = \min(U(s) - L(s), \max_{s' \in S} P_{\pi(U(s))}(s'|s)(p(s'))$$

for internal nodes. So, both BRTDP and FRTDP consider the difference between both bounds as well as the probability of transiting to the next possible states, but in a slightly different manner. Another difference is the length of the trajectories. On the one hand, BRTDP stops the current trajectory when $\sum_{s' \in S} b(s') < (U(s) - L(s)/\tau)$, where $\tau$ is a constant $> 1$ (See Lines 13 and 14 of Algorithm 5.4). On the other hand, FRTDP stops the current trajectory using an adaptive maximum depth termination. The length $D$ (see line 7 of the FRTDP function of Algorithm 5.5) of a trajectory is slightly increased by a constant $k_D$ when the states near the end of the trajectory have a greater Bellman error (difference of value of upper and lower bounds) than the other states. Finally, both approaches make a backup in a backward fashion on all visited state of a trajectory, when this trajectory has been made.

The main advantage that modified value iteration has over FRTDP and BRTDP is the pruning of the action space when $Q_U(a, s) < L(s)$. However, FRTDP and BRTDP could be easily adapted to perform this sort of pruning.

Furthermore, for a resource allocation problem, if a new task dynamically arrives in the environment, it can be accommodated by redefining the lower and upper bounds which exist at the time of its arrival, as discussed by Singh and Cohn (1998).

### 5.1.4 AND/OR Graphs based techniques

While RTDP is an on-line real-time search algorithm and AND/OR Graphs (see Appendix A for more detail) based techniques are an off-line search approach, both solve the same class of MDPs and both converge to an optimal solution without evaluating all problem states. Indeed, AO* and LAO*, which both use AND/OR graphs, compute a complete solution before setting foot in the real world. In contrast, RTDP interleaves acting and planning by generating a new state by simulating an agent executing actions, rather than by a pure computational process. We first describe the AO* Algorithm.

### AO*

Like other heuristic search algorithms, AO* (Martelli and Montanari (1978); Nilsson (1980)) can find an optimal solution without considering every problem state. Therefore, a graph is not usually supplied explicitly to the search algorithm. An implicit graph, $G$, is specified implicitly by a start state and a successor function. The search algorithm constructs an *explicit graph*, $G'$, that initially consists only of the start state. A tip or leaf state of the explicit graph is said to be terminal if it is a goal state; otherwise, it is said to be nonterminal. A nonterminal tip state can be expanded by adding to the explicit graph its outgoing $k$-connectors (one for each outcome) and any successor state(s) not already in the explicit graph.

AO* solves a state-space search problem by gradually building a solution graph, beginning from the start state. A *partial solution graph* is defined similarly to a solution graph, with the difference that the tip states of a partial solution graph may be nonterminal states of the implicit AND/OR graph. A partial solution graph is defined as follows:

- The start state belongs to a partial solution graph.
- For every non tip state in a partial solution graph, exactly one outgoing $k$-connector (corresponding to an action) is part of the partial solution graph and each of its successor states also belongs to the partial solution graph.
- Every directed path in a partial solution graph terminates at a tip state of the explicit graph.

Algorithm 5.8 outlines the AO* algorithm for finding a minimal cost solution graph in an acyclic AND/OR graph. This algorithm interleaves forward expansion of the best partial solution in Line 5 with a cost revision step that updates estimated state costs and the best partial solution in Lines 6 to 10. We also note that the best partial solution graph may have many nonterminal tip states. AO* works correctly no matter which of these tip states is chosen for expansion. However, the efficiency of AO* can be improved by using a good selection function to choose which nonterminal tip state of the best partial solution graph to expand next. Possibilities include selecting the tip state with the least estimated cost, or selecting the tip state with greatest probability of being reached. Indeed, a best-first order for expanding an AND/OR graph is to expand the state that is most likely to be part of an optimal solution is commonly used. This means expanding a tip state of the best partial solution graph.

So, in order to expand the graph in best-first order, AO* must identify the best partial solution graph in an explicit graph. To do so, AO* uses backwards induction to propagate state costs from the leaves of the explicit graph to its root. In other words,

**Algorithm 5.8** The AO* algorithm for calculating utilities of states (Nilsson, 1980).

**Function** AO*($S$)
**returns** a value function
$G' \leftarrow s_0$
**while** $G'$ has some nonterminal tip states **do**
  Expand $G'$ by choosing a nonterminal tip state such that $V(s) = h(s)$ when $s$ is not a goal state; $V(s) = 0$ otherwise
  Create $Z$ which contains the expanded state and the ancestor states from which can be reached by following the current best solution
  **repeat**
    Remove from $Z$ a state $s$ such that no descendent of $s$ in $G'$ occurs in $Z$
    $V(s) \leftarrow \min_{a \in A(s)} [C(a, s) + \gamma \sum_{s' \in S} P_a(s'|s)V(s')]$ and mark the best action for $s$
    {when determining the best action resolve ties arbitrarily, but give preference to the currently marked action}
  **until** $Z$ is empty
**end while**
**return** $G'$ (which is optimal)

AO* uses dynamic programming in its cost revision step. In its forward expansion step, it uses the start state and an admissible heuristic (optimistic values) $h(s)$ to focus computation on the part of the AND/OR graph where an optimal solution is likely to be. In summary, AO* uses branch-and-bound in its forward expansion step and dynamic programming in its cost revision step. Integrating these two techniques makes it possible to find an optimal solution as efficiently as possible, without evaluating the entire state space. AO* is an efficient off-line heuristic search algorithm for a stochastic problem representable with an acyclic graph.

### LAO*

We now describe LAO* (Hansen and Feng, 2001), a generalization of AO* that can find solutions with loops (the "L" in the name LAO* stands for "loop"). LAO* is a heuristic search algorithm that can find optimal solutions for MDPs without evaluating the entire state space. Thus, it provides a useful alternative to dynamic programming algorithms for MDPs such as value iteration and policy iteration. In fact, LAO* is the first off-line heuristic search algorithm for MDPs.

The classic AO* algorithm can only solve problems that have acyclic solutions because the backwards induction algorithm used in its cost revision step assumes an acyclic solution. The key step in generalizing AO* to create LAO* is to recognize that the cost

revision step of AO*is a dynamic programming algorithm, and to generalize this step appropriately. Instead of using backwards induction, state costs can be updated by using a dynamic programming algorithm for MDPs, in Line 7 of the algorithm, such as policy iteration or value iteration. This simple generalization sustains the algorithm LAO* summarized in Algorithm 5.9.

**Algorithm 5.9** The LAO* algorithm (Hansen and Feng, 2001).

1: **Function** LAO*($S$)
2: **returns** a value function
3: $G' \leftarrow s_0$
4: **while** $G'$ has some nonterminal tip state **do**
5: Expand $G'$ by choosing a nonterminal tip states such that $V(s) = h(s)$ when $s$ is not a goal state; $V(s) = 0$ otherwise
6: Create $Z$ which contains the expanded state and the ancestor states from which can be reached by following the current best solution
7: Solve $Z$ by performing policy iteration or value iteration
8: **end while**
9: **if** policy iteration was used **then**
10: **return** $G'$ {which is optimal}
11: **else**
12: **repeat**
13: Perform value iteration on $G'$
14: **if** the best current solution graph has an unexpanded tip state **then**
15: **go to** the **while** loop
16: **end if**
17: **until** the error bound falls below $\epsilon$
18: **return** $G'$ {which is $\epsilon$-optimal}
19: **end if**

As in AO*, the cost revision step of LAO* is only performed on the set of states that includes the expanded state and the states in the explicit graph from which the expanded state can be reached by taking marked actions (i.e., by choosing the best action for each state). The estimated cost of states that are not in this subset are unaffected by any change in the cost of the expanded state or its ancestors. The cost revision step of LAO* can be performed using either policy iteration or value iteration. An advantage of using policy iteration is that it can compute an exact cost for each state of the explicit graph after a finite number of iterations, based on the heuristic estimates at the tip states. When value iteration is used, convergence to exact state costs is asymptotic. However, this disadvantage is usually offset by the improved efficiency of value iteration for larger problems. However, the results from Bonet and Geffner

(2003b) suggest that trial-based LRTDP converge faster than LAO*.

The next section proposes tight initial bounds for RTDP with two bounds.

## 5.2 Tight Bounds for RTDP With Two Bounds

### 5.2.1 Introduction

In Section 5.1.3, we introduced RTDP with two bounds, which is nowadays the state-of-the-art in heuristic search planning in a stochastic environment. We now present an improved version of bounded RTDP (Plamondon et al. (2007c); Plamondon et al. (2007a)).

### 5.2.2 Bounded-RTDP

The Algorithm 5.10 presents a bounded version of RTDP (BOUNDED-RTDP) which aims to prune the action space of sub-optimal actions. This pruning enables to speed up the convergence and therefore to outperform LRTDP. BOUNDED-RTDP is similar to RTDP except there are two distinct initial heuristics for unvisited states $s \in S$; $h_L(s)$ and $h_U(s)$. Also, the CHECK-SOLVED($s$) procedure can be omitted because the bounds can provide the labeling of a state as solved. On the one hand, $h_L(s)$ defines a lower bound on the value of $s$ such that the optimal value of $s$ is higher than $h_L(s)$. For its part, $h_U(s)$ defines an upper bound on the value of $s$ such that the optimal value of $s$ is lower than $h_U(s)$.

The values of the bounds are computed in Lines 3 and 4 of the BOUNDED-BACKUP function. Computing these two Q-values is made simultaneously as the state transitions are the same for both Q-values. Only the values of the state transitions change. Thus, having to compute two Q-values instead of one does not augment the complexity of the approach. In fact, Smith and Simmons (2006) state that the additional time to compute a Bellman backup for two bounds, instead of one, is no more than 10%, which is also what we obtained. In particular, $L(s)$ is the lower bound of state $s$, while $U(s)$ is the upper bound of state $s$. Similarly, $Q_L(a, s)$ is the Q-value of the lower bound of action $a$ in state $s$, while $Q_U(a, s)$ is the Q-value of the upper bound of action $a$ in state $s$. Using these two bounds allow significantly reducing the action space $A$. Indeed, in Lines 5 and 6 of the BOUNDED-BACKUP function, if $Q_U(a, s) \leqslant L(s)$ then action $a$ may be pruned from the action space of $s$. In Line 13 of this function, a state can be labeled as *solved* if the difference between the lower and upper bounds is lower than $\epsilon$. When the execution goes back to the BOUNDED-RTDP function, the next state in Line 10 has a fixed number of consumable resources

available $Res_c$, determined in Line 9. In brief, PICK-NEXT-STATE($Res_c$) selects a none-*solved* state $s$ reachable under the current policy which has the highest Bellman error ($|U(s) - L(s)|$). Finally, in Lines 12 to 15, a backup is made in a backward fashion on all visited state of a trajectory, when this trajectory has been made. This strategy has been proven as efficient (Smith and Simmons (2006); McMahan et al. (2005)).

**Algorithm 5.10** The BOUNDED-RTDP algorithm (Adapted from Bonet and Geffner (2003b) and Singh and Cohn (1998)).

1: **Function** BOUNDED-RTDP($S$)
2: **returns** a value function $V$
3: **repeat**
4: $s \leftarrow s_0$
5: $visited \leftarrow null$
6: **repeat**
7: $visited.push(s)$
8: BOUNDED-BACKUP($s$)
9: $Res_c \leftarrow s.Res_c \setminus res(\pi(s))$
10: $s \leftarrow s$.PICK-NEXT-STATE($Res_c$)
{Selects a possible transiting state $s'$ with the highest $U(s') - L(s')$}
11: **until** $s$ is a goal
12: **while** $visited \neq null$ **do**
13: $s \leftarrow visited.pop()$
14: BOUNDED-BACKUP($s$)
15: **end while**
16: **until** $s_0$ is *solved* **or** $|A(s)| = 1 \ \forall \ \ s \in S$ reachable from $s_0$
17: **return** $V$

In fact, the previous algorithm 5.10 is similar to the modified value iteration proposed by Singh and Cohn (1998), described in Algorithm 5.6. In addition to the modified value iteration, BOUNDED-RTDP proposes to label states as *solved* to speed-up the convergence. Indeed, Singh and Cohn proposed that the algorithm terminates when only one competitive action remains for each state, or when the range of all competitive actions for any state are bounded by an indifference parameter $\epsilon$. Our BOUNDED-RTDP labels states for which $|U(s) - L(s)| < \epsilon$ as *solved* and the convergence is reached when $s_0$ is *solved* or when only one competitive action remains for each state. This stopping criteria is more effective since it is similar to the one used by Smith and Simmons (2006) and BRTDP by McMahan et al. (2005). Furthermore, we perform updates in a backward fashion, as done by FRTDP and BRTDP, which are proven being more effective. Thus, our BOUNDED-RTDP possesses the action pruning property of the modified value iteration, as well as the effective convergence and

**Algorithm 5.11** The bounded Bellman backup of the BOUNDED-RTDP algorithm.

1: **Function** BOUNDED-BACKUP($s$)
2: **for all** $a \in A(s)$ **do**
3: $Q_U(a,s) \leftarrow R(s) + \gamma \sum_{s' \in S} P_a(s'|s) U(s')$
4: $Q_L(a,s) \leftarrow R(s) + \gamma \sum_{s' \in S} P_a(s'|s) L(s')$
{where $L(s') \leftarrow h_L(s')$ and $U(s') \leftarrow h_U(s')$ when $s'$ is not yet visited and $s'$ has $Res_c \setminus res(a)$ remaining consumable resources}
5: **if** $Q_U(a,s) \leqslant L(s)$ **then**
6: $A(s) \leftarrow A(s) \setminus a$
7: **end if**
8: **end for**
9: $L(s) \leftarrow \max_{a \in A(s)} Q_L(a,s)$
10: $U(s) \leftarrow \max_{a \in A(s)} Q_U(a,s)$
11: $\pi(s) \leftarrow \arg\max_{a \in A(s)} Q_L(a,s)$
12: **if** $|U(s) - L(s)| < \epsilon$ **then**
13: $s \leftarrow solved$
14: **end if**

backward updates as done by FRTDP and BRTDP.

The next sections describe two separate methods to define $h_L(s)$ and $h_U(s)$. First of all, the method of Singh and Cohn (1998) is briefly described. Then, our own method (Plamondon et al. (2007c); Plamondon et al. (2007a)), which proposes tighter bounds is described in detail.

### 5.2.3 Singh and Cohn's Bounds

Singh and Cohn (1998) defined lower and upper bounds to prune the action space. Their approach is pretty straightforward. First of all, a value function is computed for all tasks to realize, using a standard RTDP approach. Then, using these *task*-value functions, a lower bound $h_L$, and upper bound $h_U$ can be defined. In particular,

$$h_L(s) = \max_{ta \in Ta} V_{ta}(s_{ta}), \text{ and } h_U(s) = \sum_{ta \in Ta} V_{ta}(s_{ta})$$

. For readability, the upper bound by Singh and Cohn is named SinghU, and the lower bound is named SinghL. The admissibility of these bounds has been proven by Singh and Cohn, such that, the upper bound always overestimates the optimal value of each state, while the lower bound always underestimates the optimal value of each

state. In this chapter, the bounds defined by Singh and Cohn and implemented using BOUNDED-RTDP define the SINGH-RTDP approach. The next sections propose to tighten the bounds of SINGH-RTDP to permit a more effective pruning of the action space.

### 5.2.4 Reducing the Upper Bound

SinghU includes actions which may not be possible to execute because of resource constraints, which outputs too high values for the upper bound. To consider only possible actions, our upper bound (Plamondon et al., 2007a), named maxU is introduced:

$$h_U(s) = \max_{a \in A(s)} \sum_{ta \in Ta} Q_{ta}(a_{ta}, s_{ta}) \tag{5.1}$$

where $Q_{ta}(a_{ta}, s_{ta})$ is the Q-value of task $ta$ for state $s_{ta}$, and action $a_{ta}$ computed using a standard LRTDP approach.

**Theorem 5.2.1** *The upper bound defined by Equation 5.1 is admissible.*

**Proof:** The local resource constraints are satisfied because the upper bound is computed using all global possible actions $a$. However, $h_U(s)$ still overestimates $V^\star(s)$ because the global resource constraint is not enforced. Indeed, each task may use all consumable resources for its own purpose. Doing this produces a higher value for each task, than the one obtained when planning for all tasks globally with the shared limited resources. ■

Computing our maxU bound in a state has a complexity of $O(|A| \times |Ta|)$. A standard Bellman backup has a complexity of $O(|A| \times |S|)$. Since $|A| \times |Ta| \ll |A| \times |S|$, the computation time to determine the upper bound of a state, which is done one time for each visited state, is much less than the computation time required to compute a standard Bellman backup for a state, which is usually done many times for each visited state. Thus, the computation time of our upper bound is negligible.

### 5.2.5 Increasing the Lower Bound

Increasing SinghL can be done by allocating the resources a priori among the tasks (Plamondon et al., 2007a). When each task has its own set of resources, each task may be solved independently. The lower bound of state $s$ is

$$h_L(s) = \sum_{ta \in Ta} Low_{ta}(s_{ta})$$

, where $Low_{ta}(s_{ta})$ is a value function for each task $ta \in Ta$, such that the resources have been allocated a priori. The allocation a priori of all the resources

is made using *marginal revenue*, which is a highly used concept in microeconomics (Pindyck and Rubinfeld, 2000), and has recently been used for coordination of a Decentralized MDP (Beynier and Mouaddib, 2006). In brief, marginal revenue is the extra revenue that an additional unit of product will bring to a firm. Thus, for a stochastic resource allocation problem, the marginal revenue of a resource is the additional expected value it involves. The marginal revenue of a resource $res$; for a task $ta$ in a state $s_{ta}$ is defined as following:

$$mr_{ta}(s_{ta}) \leftarrow V_{ta}(s_{ta}) - V_{ta}(s_{ta}(Res \setminus res)) \tag{5.2}$$

Figure 5.1 describes the processes involved in the computation of the lower bound. In the beginning, the REVENUE-BOUND function is called (Algorithm 5.12) with the set of tasks to execute and the set of available resources. Then, the ASSIGN-RESOURCE function (Algorithm 5.13) assigns each resource type to a task using the concept of marginal revenue. Finally, with the resource being shared, we compute the value function of each task with its designated set of resource. A lower bound for a state can then be obtained by summing the respective value functions.

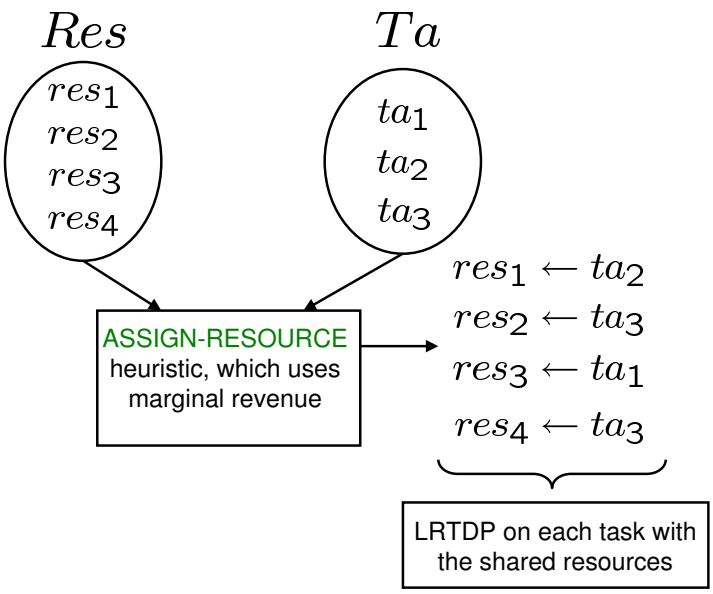



Figure 5.1: The lower bound computation process.

We now describe in more detail the Algorithm 5.12, which uses the concept of marginal revenue of a resource to allocate the resources a priori among the tasks, which enables to define the lower bound value of a state. In Line 4 of the algorithm, a value function is computed for all tasks in the environment using a standard LRTDP (Bonet and Geffner, 2003b) approach. These value functions, which are also used for the upper bound, are computed considering that each task may use all available resources. The Line 5 initializes the $value_{ta}$ variable. This variable is the estimated value of each

**Algorithm 5.12** The marginal revenue lower bound algorithm (Plamondon et al., 2007a).

1: **Function** REVENUE-BOUND($s$)
2: **returns** a lower bound $Low_{Ta}$
3: **for all** $ta \in Ta$ **do**
4: $V_{ta} \leftarrow$LRTDP($S_{ta}$) {Same value functions as used by the upper bound.}
5: $value_{ta} \leftarrow 0$
6: **end for**
7: $Res_{Ta} \leftarrow$ASSIGN-RESOURCES($S, value_{Ta}$)
8: **for all** $ta \in Ta$ **do**
9: $Low_{ta} \leftarrow$LRTDP($S_{ta}$)
10: **end for**
11: **return** $Low_{Ta}$

task $ta \in Ta$. In the beginning of the algorithm, no resources are allocated to a specific task, thus the $value_{ta}$ variable is initialized to 0 for all $ta \in Ta$.

Then, the execution shifts to the ASSIGN-RESOURCES function. In Line 5, a resource type $res$ (consumable or non-consumable) is selected to be allocated. Here, a domain expert may separate all available resources in many types or parts to be allocated. The resources are allocated in the order of its specialization. In other words, the more a resource is efficient on a small group of tasks, the more it is allocated early. Allocating the resources in this order improves the quality of the resulting lower bound. The Line 8 computes the marginal revenue of a consumable resource $res$ for each task $ta \in Ta$. For a non-consumable resource, since the resource is not considered in the state space, all other reachable states from $s_{ta}$ consider that the resource $res$ is still usable.

The approach here is to sum the difference between the real value of a state to the maximal Q-value of this state if resource $res$ cannot be used for all states in a trajectory given by the policy of task $ta$. This heuristic proved to obtain good results, but other ones may be tried, for example Monte-Carlo simulation.

In Line 18, the marginal revenue is updated in function of the resources already allocated to each task. $R(s_{g_{ta}})$ is the reward to realize task $ta$. Thus, $\frac{V_{ta}(s_{ta}) - value_{ta}}{R(s_{g_{ta}})}$ is the residual expected value that remains to be achieved, knowing current allocation to task $ta$, and normalized by the reward of realizing the tasks. The marginal revenue is multiplied by this term to indicate that, the more a task has a high residual value, the more its marginal revenue is going to be high.

Then, a task $ta$ is selected in Line 20 with the highest marginal revenue, adjusted with residual value. In Line 21, the resource type $res$ is allocated to the group of

**Algorithm 5.13** The assign resource algorithm (Plamondon et al., 2007a).

1: **Function** ASSIGN-RESOURCES($S, value_{Ta}$)
2: **returns** a lower bound $Low_{Ta}$
3: $s \leftarrow s_0$
4: **repeat**
5: $res \leftarrow$ Select a resource type $res \in Res$
6: **for all** $ta \in Ta$ **do**
7: **if** $res$ is consumable **then**
8: $mr_{ta}(s_{ta}) \leftarrow V_{ta}(s_{ta}) - V_{ta}(s_{ta}(Res \setminus res))$
9: **else**
10: $mr_{ta}(s_{ta}) \leftarrow 0$
11: **repeat**
12: $mr_{ta}(s_{ta}) \leftarrow mr_{ta}(s_{ta}) + V_{ta}(s_{ta}) - \max_{(a_{ta} \in A(s_{ta}) | res \notin a_{ta})} Q_{ta}(a_{ta}, s_{ta})$
13: $Res_{c_{ta}} \leftarrow s_{ta}.Res_{c_{ta}} \setminus res(\pi(s_{ta}))$
14: $s_{ta} \leftarrow s_{ta}$.PICK-NEXT-STATE($Res_{c_{ta}}$)
15: **until** $s_{ta}$ is a goal
16: $s \leftarrow s_0$
17: **end if**
18: $mrrv_{ta}(s_{ta}) \leftarrow mr_{ta}(s_{ta}) \times \frac{V_{ta}(s_{ta}) - value_{ta}}{R(s_{g_{ta}})}$
19: **end for**
20: $ta \leftarrow$ Task $ta \in Ta$ which maximize $mrrv_{ta}(s_{ta})$
21: $Res_{ta} \leftarrow Res_{ta} \bigcup \{res\}$
22: $temp \leftarrow \emptyset$
23: **if** $res$ is consumable **then**
24: $temp \leftarrow res$
25: **end if**
26: $value_{ta} \leftarrow value_{ta} + ((V_{ta}(s_{ta}) - value_{ta}) \times \frac{\max_{a_{ta} \in A(s_{ta}, res)} Q_{ta}(a_{ta}, s_{ta}(temp))}{V_{ta}(s_{ta})})$
27: **until** all resource types $res \in Res$ are assigned
28: **return** $Low_{Ta}$

resources $Res_{ta}$ of task $ta$. Afterwards, Line 26 recomputes $value_{ta}$. The first part of the equation to compute $value_{ta}$ represents the expected residual value for task $ta$. This term is multiplied by

$$\frac{\max_{a_{ta} \in A(s_{ta})} Q_{ta}(a_{ta}, s_{ta}(res))}{V_{ta}(s_{ta})}$$

, which is the ratio of the efficiency of resource type $res$. In other words, $value_{ta}$ is assigned to $value_{ta}$ + (*the residual value* $\times$ *the value ratio of resource type* $res$). For a consumable resource, the Q-value consider only resource $res$ in the state space, while for a non-consumable resource, no resources are available.

All resource types are allocated in this manner until *Res* is empty. All consumable and non-consumable resource types are allocated to each task. When all resources are allocated, the lower bound components $Low_{ta}$ of each task are computed in Line 9 of the REVENUE-BOUND function. When the global solution is computed, the lower bound is as follow:

$$h_L(s) = \max(\text{SINGHL}, \max_{a \in A(s)} \sum_{ta \in Ta} Low_{ta}(s_{ta})) \tag{5.3}$$

We use the maximum of the SINGHL bound and the sum of the lower bound components $Low_{ta}$, thus MARGINAL-REVENUE $\geqslant$ SINGHL. In particular, the SINGHL bound may be higher when a little number of tasks remain. As the components $Low_{ta}$ are computed considering $s_0$; for example, if in a subsequent state only one task remains, the bound of SINGHL will be higher than any of the $Low_{ta}$ components.

The main difference of complexity between SINGHL and REVENUE-BOUND is in Line 9 where a value for each task has to be computed with the shared resources. However, since the resources are shared, the state space and action space is greatly reduced for each task, reducing greatly the calculus compared to the value functions computed in Line 4 which is done for both SINGHL and REVENUE-BOUND.

**Theorem 5.2.2** *The lower bound of Equation 5.3 is admissible.*

**Proof:** $Low_{ta}(s_{ta})$ is computed with the resources being shared. Summing the $Low_{ta}(s_{ta})$ value functions for each $ta \in Ta$ does not violates the local and global resource constraints. Indeed, as the resources are shared, the tasks cannot overuse them. Thus, $h_L(s)$ is a realizable policy, and consequently an admissible lower bound. ■

### 5.2.6 Experiments and Discussion

For this problem a standard LRTDP approach has been implemented as described in Section 6.1.3. Lets summarize these approaches here:

- LRTDP-UP: The upper bound of MAXU is used for LRTDP.
- S-RTDP: The SINGHL and SINGHU bounds are used for BOUNDED-RTDP.
- M-RTDP: The REVENUE-BOUND and MAXU bounds are used for BOUNDED-RTDP.

The approaches described in this chapter are compared in Figure 5.2. To compute the lower bound of REVENUE-BOUND, all available resources have to be separated in

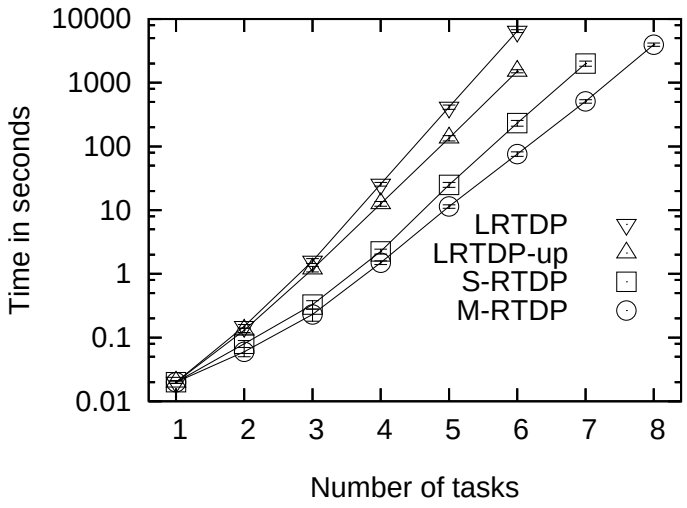



Figure 5.2: Efficiency of M-RTDP compared to S-RTDP (Plamondon et al., 2007a).

many types or parts to be allocated. For our problem, we allocated each resource of each type in the order of of its specialization like we said when describing the REVENUE-BOUND function.

In terms of experiments, notice that the LRTDP and LRTDP-UP approaches for resource allocation, which do not prune the action space, are much more complex. For instance, it took an average of 1512 seconds to plan for the LRTDP-UP approach with six tasks (see Figure 5.3). The S-RTDP approach diminished the planning time by using a lower and upper bound to prune the action space. M-RTDP further reduce the planning time by providing very tight initial bounds. In particular, S-RTDP needed 231 seconds in average to solve problem with six tasks and MR-RTDP required 76 seconds. Indeed, the time reduction is quite significant compared to LRTDP-UP, which demonstrates the efficiency of using bounds to prune the action space.

Furthermore, we implemented M-RTDP with the SinghU bound, and this was slightly less efficient than with the maxU bound. We also implemented M-RTDP with the SinghL bound, and this was slightly more efficient than S-RTDP. From these results, we conclude that the difference of efficiency between M-RTDP and S-RTDP is more attributable to the REVENUE-BOUND lower bound that to the maxU upper bound. Indeed, when the number of tasks to execute is high, the lower bounds by S-RTDP takes the values of a single task. On the other hand, the lower bound of M-RTDP takes into account the value of all task by using a heuristic to distribute the resources. Indeed, an optimal allocation is one where the resources are distributed in

the best way to all tasks, and our lower bound heuristically does that. The next section evaluates our bounds in the state of the art FRTDP and BRTDP algorithms.

### Experiments and Discussion for FRTDP and BRTDP

**Racetrack** We evaluated the performance of FRTDP (Smith and Simmons, 2006) and BRTDP (McMahan et al., 2005) on problems in the popular racetrack benchmark domain from Barto et al. (1995). We perform these experiments because FRTDP and BRTDP have never been previously compared on this problem. Foremost, we would like to compare the behavior of these algorithms for the racetrack problem and for our resource allocation problem described in the next section.

For the experiments we used the C++ code implemented by Smith and Simmons (2006). We implemented BRTDP in their simulator, which is initialized with the same (loose) initial lower and upper bounds that Smith and Simmons used. Table 5.1 reports the convergence time within $\epsilon = 10^{-3}$ for each (problem, algorithm) pair, measured both as number of backups and CPU time.

Singh and Cohn (1998) pruned the action space when $Q_U(a, s) < L(s)$. We implemented pruned versions of FRTDP and BRTDP with P-BRTDP and P-FRTDP. This pruning can be implemented efficiently with a vector of boolean, which does not require much memory, to determine if an action is dominated or not in a state. From the results, we observe that BRTDP is usually more efficient than FRTDP and the pruning slightly improve the convergence time.

Our intuition is that BRTDP is faster than FRTDP because the maximum depth termination is not well adapted for these problems, as it is the only main difference for these algorithms. We also think that the little improvement with pruning is due to the fact that the bounds are very loose and it requires many backup for the pruning to start.

**Resource Allocation** We also tested BRTDP and FRTDP in the resource allocation problem described in Section 4.7.

For this problem the FRTDP and BRTDP approaches have been implemented. For FRTDP, the initial length $D$ of a trajectory is 3 and the increasing ratio ($k_D$) is 1.2. For BRTDP the constant $\tau$ was set to 10. We tried different variations of settings and this one provided a fast convergence. Also, as Singh and Cohn (1998) proposed, we pruned the action space when $Q_U(a, s) < L(s)$ for both FRTDP and BRTDP. Lets summarize the implemented approaches here:

- LRTDP: The upper bound of MAXU is used for LRTDP.

Table 5.1: Millions of backups (CPU time) before convergence with $\epsilon = 10^{-3}$. The fastest time for each problem is shown in bold. For BRTDP, $\tau = 10$.

| Algorithm | large-b | large-b-3 | large-b-w |
|---|---|---|---|
| FRTDP | 0.30(6.07) | 0.49(7.14) | 0.85(26.81) |
| BRTDP | **0.24**(5.24) | **0.41**(5.94) | **0.74**(24.74) |
| P-FRTDP | 0.30(5.67) | 0.50(6.95) | 0.86(25.24) |
| P-BRTDP | **0.24 (4.92)** | **0.41 (5.72)** | **0.74 (23.37)** |
| Algorithm | large-ring | large-ring-3 | large-ring-w |
| FRTDP | 0.23(7.06) | 0.37(7.46) | 0.96(37.60) |
| BRTDP | **0.20** (6.87) | 0.37(7.58) | 0.84(33.97) |
| P-FRTDP | 0.23(6.62) | 0.37(**7.19**) | 0.98(34.65) |
| P-BRTDP | **0.20 (6.47)** | **0.36**(7.21) | **0.80 (31.58)** |

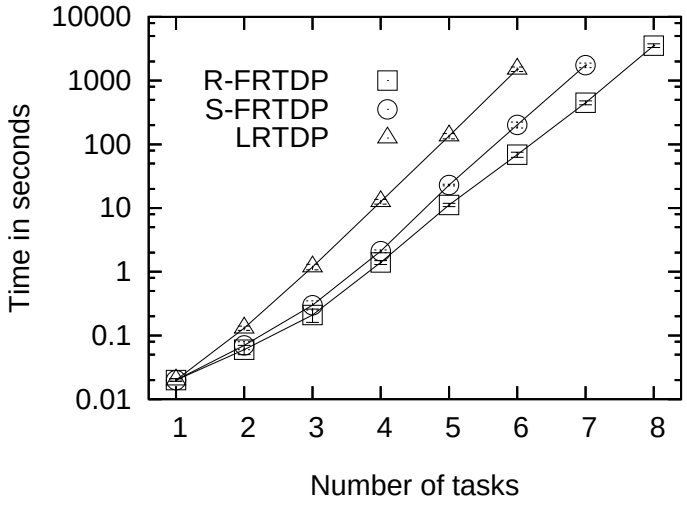



Figure 5.3: Computational efficiency of S-FRTDP, R-FRTDP and LRTDP.

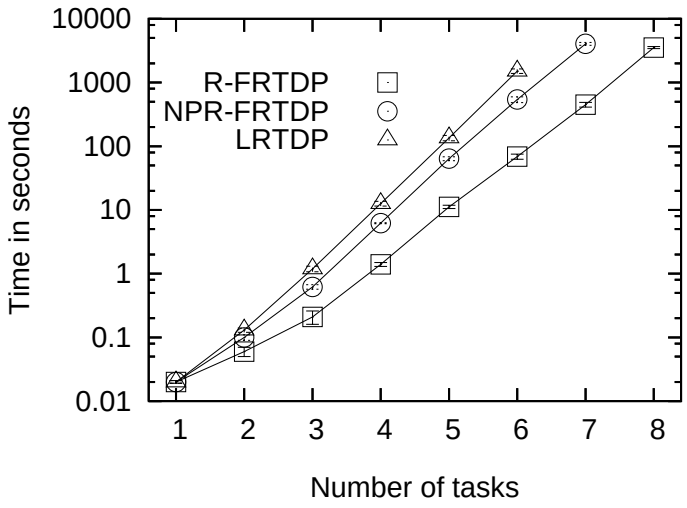



Figure 5.4: Computational efficiency of R-FRTDP, NPR-FRTDP and LRTDP.

Table 5.2: Planning time in seconds of FRTDP and BRTDP for our resource allocation problem.

| $\|Ta\|$ | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|
| S-BRTDP | 0.34 | 2.4 | 29 | 287 | 2373 | - |
| S-FRTDP | 0.3 | 2.1 | 23 | 202 | 1745 | - |
| R-BRTDP | 0.23 | 1.5 | 12.4 | 76 | 482 | 3987 |
| R-FRTDP | 0.21 | 1.4 | 11.2 | 69 | 450 | 3550 |

- S-BRTDP: The SinghL and SinghU bounds are used for BRTDP.
- S-FRTDP: The SinghL and SinghU bounds are used for FRTDP.
- R-BRTDP: The REVENUE-BOUND and maxU bounds are used for BRTDP.
- R-FRTDP: The REVENUE-BOUND and maxU bounds are used for FRTDP.
- NPR-FRTDP: R-FRTDP, but without action pruning.

To compute the lower bound of REVENUE-BOUND, all available resources have to be separated in many types or parts to be allocated. For our problem, we allocated each resource of each type in the order of of its specialization like we said when describing the REVENUE-BOUND function.

In terms of experiments, we first compared the performance of FRTDP and BRTDP on our resource allocation problem. In contrast with the racetrack problem, FRTDP was faster than BRTDP with both the Singh and Cohn bounds and our proposed bounds. Our intuition for this result is that the goal states in a resource allocation

problem has not a high depth from $s_0$. On the other hand, for a racetrack problem, the car has to traverse many states to reach the goal. In this case, the complexity of a resource allocation problem is more induced by the branching factor and the number of actions in each state. Consequently, the little trajectories's lengths of FRTDP enables to not get "lost" in the huge state space as BRTDP does. Also, the initial state receives efficient updates for FRTDP even if the trajectories's lengths are small since the goal states are usually reached, which is not the case for the racetrack problem.

Also, as one can notice in Table 5.2, and in Figures 5.3 and 5.4, the efficient trajectories of the two bounded approaches coupled with tight bounds reduce the planning time significantly. Indeed, the LRTDP approach for resource allocation, which does not prune the action space, is much more complex. For instance, it took an average of 1512 seconds to plan for the LRTDP approach with six tasks. The S-FRTDP approach diminishes the planning time by using a lower and upper bound to prune the action space and with the efficient trajectories. R-FRTDP further reduces the planning time by providing very tight initial bounds. In particular, S-FRTDP needed 202 seconds in average to solve problem with six tasks and R-FRTDP required 69 seconds. Indeed, the time reduction is quite significant compared to LRTDP, which demonstrates the efficiency of using bounds to prune the action space and produce efficient trajectories.

We may also observe on Figure 5.4 that the action space pruning was much more efficient for our resource allocation problem than it was for the racetrack problem. In average, NPR-FRTDP took 4/7 the planning time that LRTDP required to solve the same problems. With action pruning, in average R-FRTDP required only 1/12 the planning time LRTDP needed, which is a higher gain than obtained with the racetrack. Again, we explain this difference with the racetrack results by the fact that the goal states are near of $s_0$ with a resource allocation problem, and the updates permit to tighten the bounds early in the planning process, which enables to prune the action space. Also, since the initial bounds are very tight with our bounds compared with the very loose initial bounds (Smith and Simmons, 2006) used for the racetrack, pruning can be made very early.

### 5.2.7 Conclusion

This chapter presented many interesting results. First of all, the initial bounds we proposed enable a faster convergence in comparison with the Singh and Cohn (1998) bounds. These bounds have been implemented on a resource allocation problem, for which the settings have been described. Then, we compared BRTDP with FRTDP in the racetrack problem that Smith and Simmons (2006) developed. BRTDP was slightly faster than FRTDP and the action pruning enables to further diminish the planning time for both BRTDP and FRTDP.

We also implemented BRTDP and FRTDP on a resourve allocation problem. In this problem, FRTDP converges faster than BRTDP, and we argued that the low distance between the goal states with $s_0$ may explain this situation. Finally, we observed that the action pruning is very efficient for our problem. We think this is due to the tight initial bounds, and because the distance between the goal states with $s_0$ is small.

The only condition for the use of our proposed bounds is that each task possesses consumable and/or non-consumable limited resources, which we feel is a very frequent problem in Artificial Intelligence.

An interesting research avenue would be to include the time for the generation of our bounds. With a time dimension, it may be tricky to match the state of the tasks within a global state as the starting and ending time of the states may not match.

Furthermore, the heuristic search rollout algorithm (Bertsekas, 2005), which is an approximation of dynamic programming could use our marginal revenue lower bound as its initial heuristic.

Another way of reducing the complexity of a problem, is by decomposing it.

# Chapter 6

# Approach Based on Problem Decomposition

The previous chapter demonstrated how heuristic search, particulary with tight bounds, can speed up the computing time. In this chapter, we propose a different approach. We decompose the problem with the goal of reducing the computing complexity again.

In general, in the absence of any restrictions on how tasks and resources influence each other, it becomes necessary to consider the joint state and action spaces of all agents. Consequently, as the number of tasks and resources increase, the size of the MDP increases exponentially. This means that it becomes very quickly impossible to even model the problem, let alone solve it. Fortunately, in many real-world domains, the interactions between agents are not arbitrary. In this chapter, we focus on decomposition techniques where the interactions between tasks and resources have particularities, which permits to diminish the problem complexity. The following section proposes to use Q-decomposition for a specific resource allocation problem.

## 6.1 A Q-decomposition Approach

### 6.1.1 Introduction

To solve efficiently problems where the resources are already shared among the agents, but the actions made by an agent may influence the reward obtained by at least another agent, we adapt Q-decomposition proposed by Russell and Zimdars (2003). In our Q-decomposition approach (Plamondon et al. (2006c); Plamondon et al. (2007b); Plamondon et al. (2007a)), a planning agent manages each task and all agents have to share the limited resources. The planning process starts with the initial state in

which each agent computes their respective Q-values. Then, the planning agents are coordinated through an arbitrator to find the highest global Q-value by adding the respective possible Q-values of each agent. We implemented Q-decomposition in a heuristic search approach. Since the number of states and actions to consider when computing the optimal policy is exponentially reduced with Q-decomposition, compared to other known approaches, it allows to formulate the first optimal decomposed heuristic search algorithm in a stochastic environments.

### 6.1.2 Q-decomposition for Resource Allocation

There can be many types of resource allocation problems, as represented in Figure 6.1. Firstly, if the resources are already shared among the agents (Problem 1), and the actions made by an agent does not influence the state of another agent, the globally optimal policy can be computed by planning separately for each agent.

A second type of resource allocation, represented in Problem 2, is where the resources are already shared among the agents, but the actions made by an agent may influence the reward obtained by at least another agent. For instance, a group of agents which manages the oil consummated by a country falls in this group. These agents desire to maximize their specific reward by consuming the right amount of oil. However, all the agents are penalized when an agent consumes oil because of the pollution it generates. Another example of this type comes from our problem of interest, explained in Section 4.7, where, in some scenarios, it may happens that the missiles can be classified in two types: Those requiring a set of resources $Res_1$ and those requiring a set of resources $Res_2$. This can happen depending on the type of missiles, their range, and so on. In this case, two agents can plan for both set of tasks to determine the policy. However, there are interactions between the resource of $Res_1$ and $Res_2$, so that certain combination of resource cannot be assigned. In particular, if an agent $i$ allocate resources $Res_i$ to the first set of tasks $Ta_i$, and agent $i'$ allocate resources $Res_{i'}$ to second set of tasks $Ta_{i'}$, the resulting policy may include actions which cannot be executed together. The third type of problem is where all resources are available to all tasks and no Q-decomposition is possible. Still, heuristic search (Chapter 5), acyclic decomposition (Section 6.2), and MTAMDPs (Section 6.3) can be used to diminish the planning time in this case.

To resolve conflicts between resources, we use Q-decomposition proposed by Russell and Zimdars (2003) in the context of reinforcement learning. The primary assumption underlying Q-decomposition is that the overall reward function $R$ can be additively decomposed into separate rewards $R_i$ for each distinct agent $i \in Ag$, where $|Ag|$ is the number of agents. That is, $R = \sum_{i \in Ag} R_i$. It requires each agent to compute a value, from its perspective, for every action. To coordinate with each other, each agent $i$ reports its action values $Q_i(a_i, s_i)$ for each state $s_i \in S_i$ to an arbitrator at each learn-

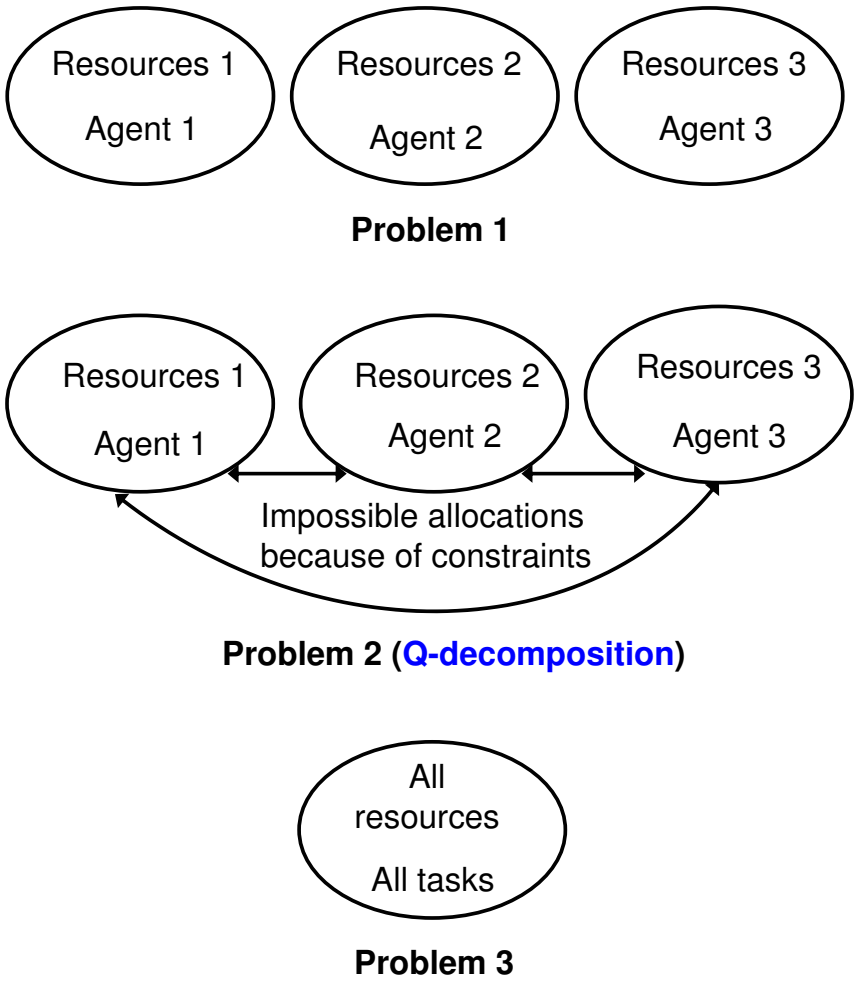



Figure 6.1: Different types of resource allocation problem.

ing iteration. The arbitrator then chooses an action maximizing the sum of the agent Q-values for each global state $s \in S$. The next time state $s$ is then updated, an agent $i$ considers its Q-value as its respective contribution to the global maximal Q-value. That is, $Q_i(a_i, s_i)$ is the value of a state such that it maximizes $\max_{a \in A(s)} \sum_{i \in Ag} Q_i(a_i, s_i)$. The fact that the agents use a determined Q-value as the value of a state is an extension of the Sarsa on-policy algorithm (Rummery and Niranjan, 1994) to Q-decomposition. Russell and Zimdars called this approach local Sarsa. In this way, an ideal compromise can be found for the agents to reach a global optimum. Indeed, rather than allowing each agent to choose the successor action, each agent $i$ uses the action $a'_i$ executed by the arbitrator in the successor state $s'_i$:

$$Q_i(a_i, s_i) = R_i(s_i) + \gamma \sum_{s'_i \in S_i} P_{a_i}(s'_i|s_i) Q_i(a'_i, s'_i) \tag{6.1}$$

where the remaining consumable resources in state $s'_i$ are $Res_{c_i} \setminus res_i(a_i)$ for a resource allocation problem. Russell and Zimdars (2003) demonstrated that local Sarsa converges to the optimum. In addition, this form of agent decomposition allows in some cases the local Q-functions to be expressed by a much reduced state and action space.

Figure 6.2 gives an example of Q-decomposition. The two agents, dollars and euros, start in state $s_0$ and can attempt to move $Left$, $Up$, or $Right$, or it can stay put ($NoOp$). The agents have to be coordinated so that they execute the same action at the same time. Also, they have to execute an action until a goal state is reached. With probability $\epsilon$, each action has no effect; otherwise, the agents reach a terminal state with rewards of dollars and/or euros as shown. If we assume rough parity between dollars and euros, then the optimal policy is clearly to go $Up$. The question is how to achieve this with a distributed architecture in which one agent cares only for dollars and the other only for euros. Suppose the case where $\epsilon = 0.2$ and $\gamma = 0.95$, and that the two agents plan on their own. After they have converged, they send their Q-values to the arbitrator which selects the global action which maximizes the Q-value of all action combination for the agents. For this case, the Q-values computed by the arbitrator are in Table 6.1. The optimal action in this scenario is the $NoOp$ action. Indeed, by doing nothing, the agents are penalized by the discount factor, afterwards they can execute their optimal action ($Left$ for dollars and $Right$ for euros) in the subsequent state. However, the $NoOp$ action is by far the worse action, because the agents will obtain no money. All other actions are by far better. Furthermore, the optimal action, which is to go $Up$ has a significantly lower Q-value than the $NoOp$ action.

On the other hand, if the agents use Q-decomposition, the agents are able to converge to the optimal. Indeed, Table 6.2 details the process of the first iteration of Q-decomposition. In this first iteration, the arbitrator computes that the $Up$ action has the highest Q-value. In the next iteration of the planning process, the agents will compute 0.48 for the value of state $s_0$, and not 0.8, which they would have by planning

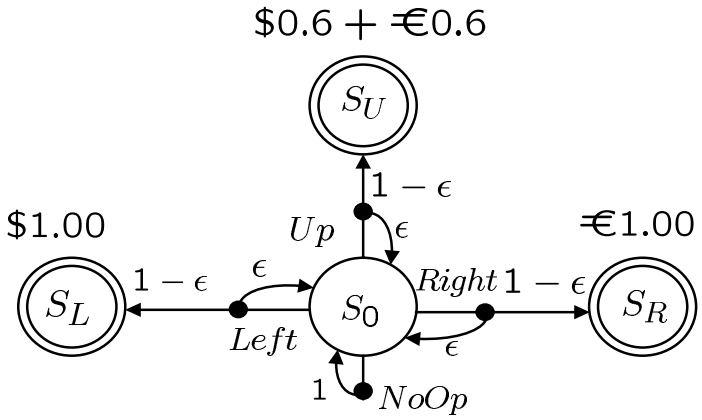



Figure 6.2: Problem type for Q-decomposition (adapted from Russell and Zimdars (2003)).

Table 6.1: Q-values for the agents planning independently with the problem of Figure 6.2, where $\epsilon = 0.2$ and $\gamma = 0.95$.

| Action | Value |
|---|---|
| *Left* | \$0.988 and €0.188 = 1.176 |
| *Right* | \$0.188 and €0.988 = 1.176 |
| *Up* | \$0.63 and €0.63 = 1.26 |
| *NoOp* | \$0.93 and €0.93 = 1.86 |

independently. In this problem, the Q-decomposition permits for the agents to converge to the globally optimal action, which is to go $Up$.

Table 6.2: Q-values for the agents using Q-decomposition at the first iteration with the problem of Figure 6.2, where $\epsilon = 0.2$ and $\gamma = 0.95$.

| Action | Value |
|---|---|
| $Left$ | \$0.8 and €0 = 0.8 |
| $Right$ | \$0 and €0.8 = 0.8 |
| $Up$ | \$0.48 and €0.48 = 0.96 |
| $NoOp$ | \$0 and €0 = 0 |

For our resource allocation problem described in this section, Q-decomposition can be applied to generate an optimal solution. Indeed, an optimal Bellman backup can be applied in a state as in Algorithm 6.1. In Line 5 of the QDEC-BACKUP function, each agent managing a task computes its respective Q-value. Here, $Q_i^\star(a_i', s')$ determines the optimal Q-value of agent $i$ in state $s'$. An agent $i$ uses as the value of a possible state transition $s'$ its Q-value which is part of the maximal global Q-value for state $s'$ as in the original Q-decomposition approach. In brief, for each visited state $s \in S$, each agent computes its respective Q-values with respect to the global state $s$. So the state space is the joint state space of all agents. Some of the gain in complexity to use Q-decomposition resides in the $\sum_{s_i' \in S_i} P_{a_i}(s_i'|s)$ part of the equation. An agent considers as a possible state transition only the possible states of the set of tasks it manages. Since the number of states is exponential with the number of tasks, using Q-decomposition should reduce the planning time significantly. Furthermore, the action space of the agents takes into account only their available resources which is much less complex than a standard action space, which is the combination of all possible resource allocation in a state for all agents.

The arbitrator functionalities are depicted in Lines 8 to 20. The global Q-value is the sum of the Q-values produced by each agent managing each task as shown in Line 11, considering the global action $a$. In this case, when an action of an agent $i$ cannot be executed simultaneously with an action of another agent $i'$, the global action is simply discarded from the action space $A(s)$. Line 14 simply allocates the current value with respect to the highest global Q-value, as in a standard Bellman backup. Then, the optimal policy and Q-value of each agent is updated in Lines 16 and 17 to the sub-actions $a_i$ and specific Q-values $Q_i(a_i, s)$ of each agent for action $a$.

The behavior of QDEC-BACKUP is now discussed and we start it by proving the optimality of QDEC-BACKUP. This proof requires to demonstrate that the QDEC-BACKUP function outputs the same optimal value as a standard Bellman backup.

**Algorithm 6.1** The Q-decomposition Bellman Backup algorithm (Plamondon et al., 2007a).

1: **Function** QDEC-BACKUP($s$)
2: $V(s) \leftarrow 0$
3: **for all** $i \in Ag$ **do**
4: **for all** $a_i \in A_i(s)$ **do**
5: $Q_i(a_i, s) \leftarrow R_i(s) + \gamma \sum_{s'_i \in S_i} P_{a_i}(s'_i|s) Q^\star_i(a'_i, s')$
{where $Q^\star_i(a'_i, s') = h_i(s')$ when $s'$ is not yet visited, and $s'$ has $Res_{c_i} \setminus res_i(a_i)$ remaining consumable resources for each agent $i$}
6: **end for**
7: **end for**
8: **for all** $a \in A(s)$ **do**
9: $Q(a, s) \leftarrow 0$
10: **for all** $i \in Ag$ **do**
11: $Q(a, s) \leftarrow Q(a, s) + Q_i(a_i, s)$
12: **end for**
13: **if** $Q(a, s) > V(s)$ **then**
14: $V(s) \leftarrow Q(a, s)$
15: **for all** $i \in Ag$ **do**
16: $\pi_i(s) \leftarrow a_i$
17: $Q^\star_i(a_i, s) \leftarrow Q_i(a_i, s)$
18: **end for**
19: **end if**
20: **end for**

**Lemma 6.1.1** *A state for QDEC-BACKUP is updated in the same manner as for a standard Bellman backup.*

**Proof:** The following equation is used in QDEC-BACKUP to compute a Q-value:

$$Q(a, s) = \sum_{i \in Ag} R_i(s) + \max_{a_i \in A_i(s)} \gamma \sum_{s'_i \in S'_i} P_{a_i}(s'_i|s) Q^\star_i(a'_i, s') \tag{6.2}$$

Since the reward can be additively decomposed for each task, Equation 6.2 may be rewritten as:

$$Q(a, s) = R(s) + \sum_{i \in Ag} \max_{a_i \in A_i(s)} \gamma \sum_{s'_i \in S'_i} P_{a_i}(s'_i|s) Q^\star_i(a'_1, s') \tag{6.3}$$

Since $Q(a, s) = \sum_{i \in Ag} Q_i(a, s)$ when the transition probability of each task considers the actions performed on other tasks, Equation 6.3 may be rewritten as:

$$Q(a, s) = R(s) + \max_{a \in A(s)} \gamma \sum_{s' \in S'} P_a(s'|s) Q(a', s') \tag{6.4}$$

where $Q(a', s')$ is the maximal Q-value for state $s'$. Indeed, since the arbitrator determines the maximal Q-value for a state, $Q(a', s') = V(s')$. Since Equation 6.4 is the same as a Bellman backup, and $Q(a', s')$ is the same as $V(s)$, a Q-value is updated in the same manner in QDEC-BACKUP as for a standard Bellman backup. ■

A standard Bellman backup has a complexity of $O(|A| \times |S_{Ag}|)$, where $|S_{Ag}|$ is the number of joint states for all agents excluding the resources, and $|A|$ is the number of joint actions. On the other hand, the Q-decomposition Bellman backup has a complexity of $O((|Ag| \times |A_i| \times |S_i)|) + (|A| \times |Ag|))$, where $|S_i|$ is the highest number of states for an agent $i$, excluding the resources, and $|A_i|$ is the highest number of actions for an agent $i$. Since $|S_{Ag}|$ is combinatorial with the number of tasks, so $|S_i| \ll |S|$. Also, $|A|$ is combinatorial with the number of resource types. If the resources are already shared among the agents, the number of resource types for each agent will usually be lower than the set of all available resource types for all agents. In these circumstances, $|A_i| \ll |A|$. In a standard Bellman backup, $|A|$ is multiplied by $|S_{Ag}|$, which is much more complex than multiplying $|A|$ by $|Ag|$ with the Q-decomposition Bellman backup. Thus, the Q-decomposition Bellman backup is much less complex than a standard Bellman backup. Furthermore, the communication cost between the agents and the arbitrator is null since this approach does not consider a geographically separated problem.

### 6.1.3 Experiments and Discussion

The domain of the experiments is as described in Section 4.7. For the Q-decomposition, the number of resource types has been fixed to 5, where there are 3 consumable resource types and 2 non-consumable resources types.

First, we implemented a standard LRTDP approach in which, a simple heuristic has been used where the value of an unvisited state is assigned as the value of a goal state such that all tasks are achieved. This way, the value of each unvisited state is assured to overestimate its real value since the value of achieving a task $ta$ is the highest the planner may get for $ta$. Since this heuristic is pretty straightforward, the advantages of using better heuristics are more evident. Nevertheless, even if the LRTDP approach uses a simple heuristic, still a huge part of the state space is not visited when computing the optimal policy.

We also implemented a QDEC-LRTDP approach where the backups are computed using the QDECBACKUP function (Algorithm 6.1), and using the LRTDP algorithm. In particular; the updates made in the CHECK-SOLVED function are also made using the QDEC-BACKUP function. To implement QDEC-LRTDP, we divided the set of tasks in two equal parts. The set of tasks $Ta_i$, managed by agent $i$, can be accomplished with the set of resources $Res_i$, while the second set of tasks $Ta_{i'}$, managed by agent $Ag_{i'}$, can be accomplished with the set of resources $Res_{i'}$. $Res_i$ had one consumable resource

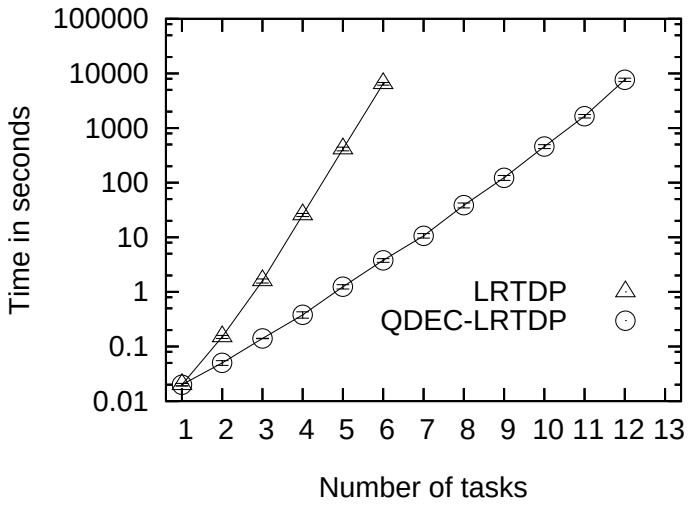



Figure 6.3: Efficiency of Q-decomposition LRTDP (QDEC-LRTDP) and LRTDP (Plamondon et al., 2007a).

type and one non-consumable resource type, while $Res_{i'}$ had two consumable resource types and one non-consumable resource type. When the number of tasks is odd, one more task is assigned to $Ta_{i'}$. There are constraint between the group of resources $Res_i$ and $Res_{i'}$ such that some assignments are not possible. These constraints are managed by the arbitrator as described in Section 6.1.

The approaches described in this section are compared in Figure 6.3. As we can see, the Q-decomposition permits to diminish the planning time significantly in our problem settings. In addition , it seems to be a very efficient approach when a group of agents have to allocate their respective resources where the action made by an agent may influence the reward obtained by at least another agent.

Our experiments have shown that QDEC-LRTDP (Plamondon et al. (2006c); Plamondon et al. (2007b); Plamondon et al. (2007a)) provides a potential solution to solve efficiently stochastic resource allocation problems. Indeed, the theoretical and experimental complexities of QDEC-LRTDP are significantly lower than for LRTDP. While the discussion in this thesis focusses on resource allocation problems, QDEC-LRTDP may be used in any type of problem where the overall reward function can be additively decomposed into separate rewards for distinct agents. Indeed, the general principle of the original Q-decomposition approach has to be applicable to justify using QDEC-LRTDP.

An interesting research avenue would be to experiment Q-decomposition with other heuristic search algorithms than LRTDP. HDP (Bonet and Geffner, 2003a), and LAO$^\star$ (Hansen and Feng, 2001) are both efficient heuristic search algorithms and may be

greatly improved if combined with Q-decomposition. However, when the resources are available to all agents, no Q-decomposition is possible. In this case, Bounded Real-Time Dynamic Programming (BOUNDED-RTDP), presented in Section 5.2 permits to focuss the search on relevant states, and to prune the action space $A$ by using lower and higher bound on the value of states.

The next section describes a decomposition made to the MDP model induced by a specific particularity of our problem: Some tasks may create other tasks.

## 6.2 Decomposition for a Loosely-Coupled Resource Allocation Problem

### 6.2.1 Introduction

There is an explosion of the state space in our type of problem, due to Bellman's curse of dimensionality (Bellman, 1957). To alleviate this, many researchers decomposed the state space to diminish the planning time (Meuleau et al. (1998); Dolgov and Durfee (2004); Wu and Castanon (2004)).

In this section, we focus on resource allocation problems where the interactions between tasks are localized (each task only affects a small number of neighbors). Central to the model that we use in this section is the concept of a dependency graph that describes the relationships between agents. The idea is very similar to other graphical models, e.g., graphical games (Kearns et al., 2001), coordination graphs (Guestrin et al., 2003), and multiagent influence diagrams (Koller and Milch, 2003), where graphs are used to more compactly represent the interactions between agents to avoid the exponential explosion in problem size. Similarly, our representation can be exponentially smaller than the size of the flat MDP defined on the joint state and action spaces of all agents. A distinguishing characteristic of the graphical representation that we use is that it makes more fine-grained distinctions about how agents affect each other: the problem is decomposed in loosely connected components in the presence of tasks which may influence the presence of other tasks. However, the states of a task are strongly connected since all non-absorbing states are always reachable from each other.

We present below two novel decomposition techniques which solve efficiently the problem when the strongly connected states are grouped in separate components (Plamondon et al., 2005). These techniques are a specialization of the Markov Decision Process (MDP) framework, associated to our problem characteristics. Another originality of our work is to consider effectively the criticality of each task, by defining a weight factor. We now formalize the problem.

### 6.2.2 Formulation of Loosely-Coupled Tasks Problem

We consider a state as a conjunction of tasks to accomplish by a *planning agent*. A task corresponds to a work to accomplish for an agent. On the other hand, an action corresponds to use a resource to accomplish a task. An action may change the probability to execute a particular task, or all tasks. The tasks may be in many possible states, which are strongly connected. The states are strongly connected because from a non-absorbing state $s$ in a task, we can transit to any other non-absorbing state $s'$ of this task, and go back to $s$ (Tarjan, 1972). Furthermore, we have some tasks which may be created by another task. We consider the creation of a task as a stochastic process. A task which may not create another task is considered as *critical*. The parents of a critical task, considered as *non-critical*, have to be accomplish, because they may create a critical task.

Figure 6.4 gives an example of the type of problem we are interested in this section. On the figure, there are two possible tasks to accomplish : $ta_1$ and $ta_2$. Suppose the problem begins in state $s_1$. The states of task $ta_1$ are strongly connected because from a non-absorbing state $s$ in $ta_1$, we can transit to any other non-absorbing state $s'$ of this task, and go back to $s$. Task $ta_1$ is non-critical, since it may create $ta_2$. When the critical task $ta_2$ is created, we cannot go back to the states of task $ta_1$. Thus, the states of a task are strongly connected, but a task can only be weakly connected to another tasks.

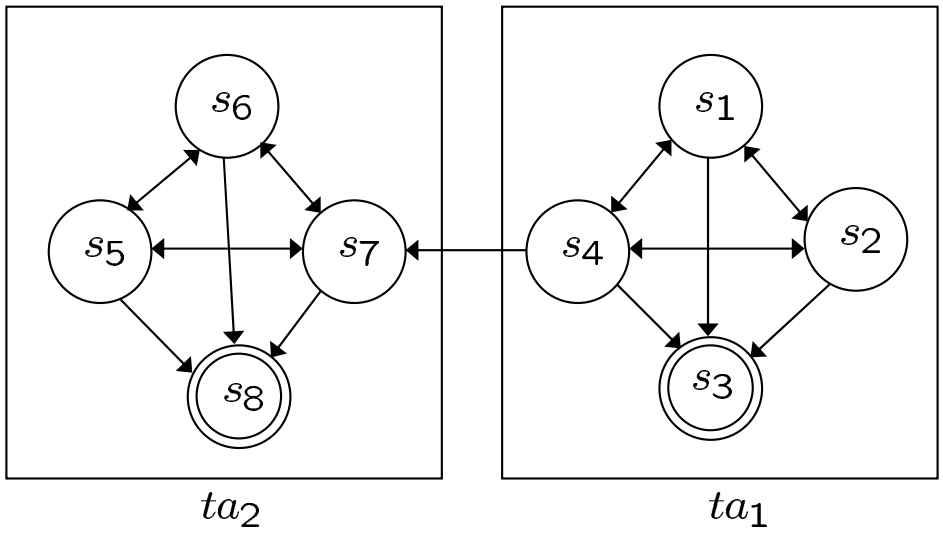



Figure 6.4: Two tasks for which states are strongly connected. $ta_1$ is a non-critical task, while $ta_2$ is a critical task.

Furthermore, the planning agent is under hard real-time constraints to produce an effective policy. The very high number of states in this type of problem coupled with

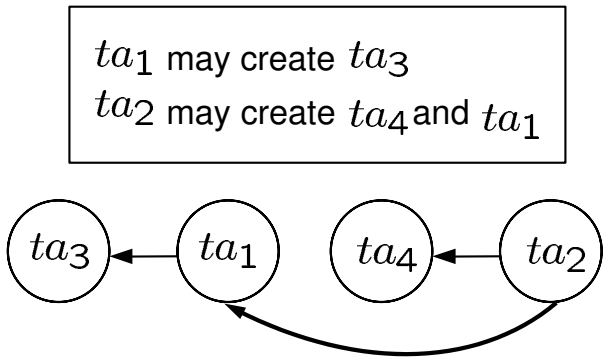



Figure 6.5: Acyclic graphs of task dependencies.

the time constraint makes it very complex.

We should note that a resource allocation problem is known as being NP-complete (Zhang, 2002). The state space consists of the cross product of each task individual state spaces and the available resources; the action space is the set of resource assignment. Meuleau et al. (1998) proposed an approach where each task is solved optimally in an independent manner. The solutions are merged afterwards with a greedy strategy, thus producing an approximate solution. On the other hand, Dolgov and Durfee (2004) extended Meuleau's work by assigning resources to each task and then producing the global optimal policy. Another promising approach is to plan for the resources separately (Wu and Castanon, 2004). In particular, the approaches of all cited authors do not consider the fact that a task may create other tasks, which is the base of our algorithms.

### Task Weighting

We have developed an heuristic to weight the importance of each task; based on the fact that some tasks are dependent on others in the sense where they may create them. We can represent these dependencies between tasks with a graph. An important characteristic for the problem we are interested in, is that the representation of task dependencies in a graph produces a certain number of acyclic graphs. Note that an acyclic graph is one that all state transitions always results in a state that were never previously visited, thus implying a partial order on the set of states. Figure 6.5 represents the acyclic graphs where two tasks, $ta_1$ and $ta_2$, are in the environment. $ta_1$ may create task $t_3$, and $ta_2$ may create task $ta_4$. In our problem setting, task $ta_3$ and $ta_4$, which are leafs, are the only ones that are necessary to execute. The non-critical tasks have to be executed only because they can create critical tasks. We use Algorithm 6.2 to define the weight factor $W(ta)$ representing the relative importance to accomplish each task $ta$.

**Algorithm 6.2** The task weighting algorithm (Plamondon et al., 2005).

1: **Function** TASK-WEIGHTING($DepG$)
2: **while** $DepG \neq null$ **do**
3: Remove from $DepG$ a task $ta$, such that no descendent of $ta$ is in $DepG$
4: **if** $ta$ is a leaf task **then**
5: $W(ta) \leftarrow P(Fail(ta)|s_{ta}) \times \text{CONS}(ta)$
6: **else**
7: $W(ta) \leftarrow \sum_{ta' \in |Ta|} P(\text{C}(ta')|s_{ta}) \times E(ta'|(\text{C}(ta')|s_{ta})) \times W(ta')$
8: **end if**
9: **end while**

The $DepG$ graph contains many acyclic graphs, which nodes (group of tasks) are affected a weight factor. As seen in Line 3 of the algorithm, the tasks are traversed using a backward approach just like in the AO* algorithm (Nilsson, 1980) for an acyclic graph. For a critical task $ta_i$, we define the weight factor $W(ta_i)$ as the product of the probability that the task is failed of being accomplished by the planning agent, considering its current state ($P(Fail(ta)|s_{ta})$), by the consequence it may inflict to the planning agent ($\text{CONS}(ta)$). For a non-critical (non-critical) task, the weight is specified according to the sum of each task $ta'$, as the product of the probability of creating task ($P(\text{C}(ta')|s_{ta})$), by the expected number of created tasks $ta'$ ($E(ta'|(\text{C}(ta')|s_{ta}))$), and by the weight factor of $ta'$ ($W(ta')$). The probabilities $P(\text{C}(ta')|s_{ta})$ and $P(Fail_{ta}|s_{ta})$ are known a priori considering that no action is made. Consequently, the weight function overestimates the weight of all tasks, since the probability of creating other tasks is higher when no action is made, than with an optimal policy. However, the weight of a task should not change as the computing of a policy is made, because some state value may be overestimated. Considering this, we have used this approach to define the probability, which is an approximation. The next section describes two decomposition strategies to reduce the planning complexity.

### 6.2.3 Decomposition Techniques

#### Acyclic Decomposition Algorithm

An efficient decomposition technique may regroup together strongly connected states (Meuleau et al., 1998). In the same sense, it is known that a planning problem which may be represented with an acyclic graph, instead of a cyclic one, is generally easier to solve (Hansen and Feng, 2001). We recall that an acyclic graph is one that all state transitions always results in a states that were never previously visited, thus implying a partial order on the set of states. On the other hand, a cyclic graph may visit certain

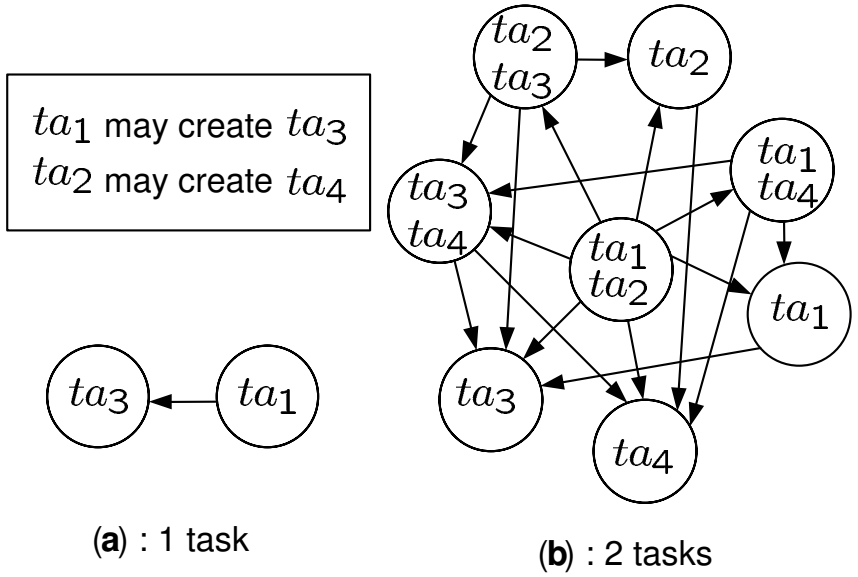



Figure 6.6: The acyclic graph of cyclic components.

states more than one time. The idea we have here is to transform our problem in an abstract acyclic one, which contains many cyclic components (i.e. states that may transit to each other many times). A component corresponds to the states of a group of tasks, and the graph contains a component for each possible task combination. Since each state of a task can transit to each other many times, the components are cyclic. On the other hand, the acyclic graph represents the possible task transitions. One may use Tarjan (1972) linear algorithm for detecting the *strongly-connected components* of a directed graph to create the acyclic graph. Figure 6.6 shows the acyclic graph when task $ta_1$ (a) or task $ta_1$ and $ta_2$ (b) of Figure 6.5 are in the environment. All nodes represent a cyclic component. This graph supposes that a task may create one other task in maximum. The significance of a link in Figure 6.6, means simply a change in the tasks the planning agent has to achieve. Thus, it has a task transition meaning, as opposed to a task creation in Figure 6.5. Once the abstract acyclic graph is formed, we can solve it using a backward approach just like in the AO* algorithm (Nilsson, 1980). Algorithm 6.3 describes how we calculate the value $V_c$ of each strongly connected component $c$ of an acyclic graph $AcG$.

**Algorithm 6.3** The acyclic decomposition algorithm (Plamondon et al., 2005).

1: **Function** ACYCLIC-DEC($AcG$)
2: **while** $AcG \neq null$ **do**
3: Remove from $AcG$ a component $c$, such that no descendent of $c$ is in $AcG$
4: $V_c \leftarrow$MDP-ALGO($c$)
5: **end while**

This algorithm solves each component, using an MDP algorithm ("MDP-ALGO($c$)"), from the leaf to the root of the graph. This way, each component may only transit to a solved component, thus each component have to be solved one time. "MDP-ALGO($c$)" may be any algorithm to solve an MDP, such as standard approaches like value iteration or policy iteration. For example, in Figure 6.6 (b), in the first iteration, we can choose to remove whichever of $ta_3$ or $ta_4$, and solve the component using MDP-ALGO($c$). Then, if we removed $ta_3$ in the first iteration, we may now remove $ta_4$, or vice versa. When both $ta_3$ and $ta_4$ are removed, $ta_1$ and $ta_2$ may be removed. We remove components in this order, until the component $ta_1, ta_2$ is removed and solved.

**Theorem 6.2.1** *The acyclic decomposition algorithm produces an optimal policy.*

**Proof:** It is known that the AO* algorithm generates an optimal policy of a problem represented by an acyclic graph (Nilsson, 1980). However, the nodes in our graph regroups many states (component), while in AO*, it is an atomic state. Thus, to prove the optimality of the acyclic decomposition algorithm, we have to determine whether a component exhibits the same characteristics as a state. The action of all states in the component are optimal since the they are computed using an optimal MDP algorithm. Then, we could say that the component, which is an abstract state, has an abstract optimal action. Thus, the computation of the optimal action for a state in AO*, may be viewed as the computation of the optimal abstract action for a component in our algorithm. It follows that the acyclic decomposition algorithm produces an optimal policy. ■

The planning time is reduced compared to a standard approach, however, as we can observe on Figure 6.7 of Section 6.2.4, it still takes about 20 seconds to plan for a problem with three initial tasks. For our hard real-time problem, we should further reduce the complexity of the planner. To this end, the next section introduces an on-line decomposition strategy to further reduce the planning time; at the compromise, however, of producing an approximate solution.

### On-line Decomposition Algorithm

A standard dynamic programming (i.e. value iteration or policy iteration), or the previous acyclic decomposition approach to solve the problem is done off-line. Indeed, all the computation is made before executing any action. Since we need a decision in real-time, we introduce an on-line approach, in which planning and the execution of actions are carried out concurrently. This approach is elaborated based on the acyclic decomposition approach. Indeed, the approach plans a separate policy for each task (i.e. component with one task), using the ACYCLIC-DEC(graph $AcG$) function. A solution for one task is produced very quickly using the acyclic algorithm. We adopt

this approach to plan for each task, because each task are strongly connected, but loosely connected with the other tasks. Algorithm 6.4 overviews our decomposition approach.

**Algorithm 6.4** The on-line decomposition algorithm (Plamondon et al., 2005).

1: **Function** ON-LINE-DEC($S$)
2: **for all** task $ta \in |Ta|$ **do**
3: $V_{ta} \leftarrow$ ACYCLIC-DEC($S_{ta}$)
4: **end for**
5: **repeat**
6: **if** the state of a task has changed **then**
7:
$$\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{ta \in Ta} R(S_{ta}) + \sum_{s'_{ta} \in S_{ta}} P_{a_{ta}}(s'_{ta}|S_{ta}) Q_{ta}(a_{ta}, s'_{ta})$$
8: **end if**
9: **until** the scenario is over

The first part of the algorithm computes a value function $V_{ta}$ for the state space of each task $S_{ta}$ in the environment using a standard dynamic programming approach ("ACYCLIC-DEC($S_{ta}$)"). For example, if one task is in the environment, we compute the value function for this task only, and all of its potential descendants. When the component value function of each task is made, we proceed to the on-line phase. In particular, we determine the global sub-optimal action to execute each time a task changes. The local and global resource constraints on the amount that can be used had been verified in the off-line planning of each task. When computing the global action, we verify the local and global constraint for this state only, thus producing an approximative solution. We now detail the results we have obtained using three algorithm; standard dynamic programming, the acyclic decomposition as presented by Algorithm 6.3 of Section 6.2.3, and the on-line decomposition as presented by Algorithm 6.4 of Section 6.2.3.

### 6.2.4 Experiments and Discussion

We tested the problem on the domain described in Section 4.7. A difference from the initial domain is that $|S_{ta}| = 10$, thus each task can be in ten distinct states. We computed a policy with a certain number of initial tasks which may creates other tasks.

From the results, as shown on Figure 6.7, we can conclude that planning an optimal policy, using a standard dynamic programming approach (value iteration), for this problem is very complex. The acyclic decomposition algorithm reduces the planning

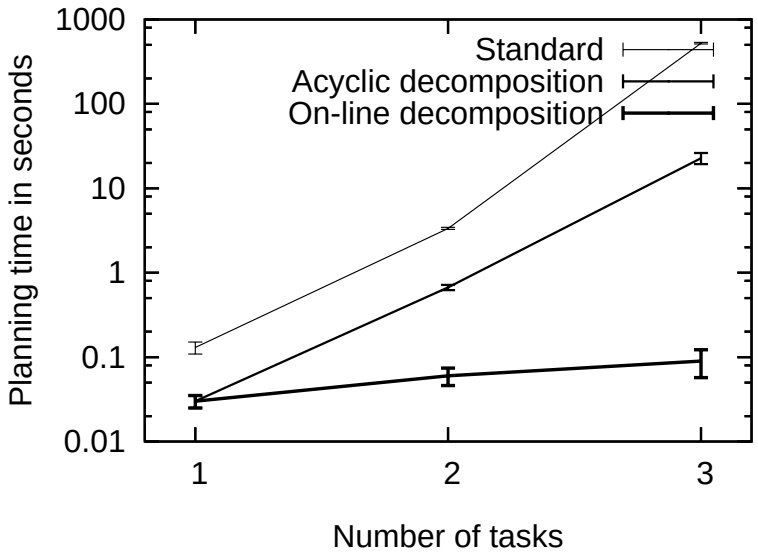



Figure 6.7: Planning time for different cases (Plamondon et al., 2005).

time, but it is still too high for our real-time application. So, this approach is promising, but we should improve it in the future. On the other hand, the on-line decomposition algorithm produces an approximative policy in a very short time.

Table 6.3: Number of times that we obtain the "optimal" with our on-line decomposition planner (Plamondon et al., 2005).

| # of tasks | Average cases | Worst cases |
|---|---|---|
| 1 | 100 $\pm 0\%$ | 100 |
| 2 | 99,94 $\pm 0,035\%$ | 99,90 |
| 3 | 99,91 $\pm 0,042\%$ | 99,85 |

Table 6.3 details the percentage of the optimal obtained with our on-line decomposition planner. We can see that the value of the policy decreases when the number of tasks augment. As a matter of fact, when one threat is present, the policy is optimal, because it uses the optimal acyclic decomposition algorithm for the task.

To sum up, we can note that, we are interested by a multi-tasks problem, where the presence of some tasks is dependent on the presence of others. Such problem produces an enormous state space. To compute a policy in a timely manner for this sort of problem, we have decomposed the state space. In particular, a task is strongly coupled within its own possible states, but weakly coupled with the other tasks. In our approach (Plamondon et al., 2005), we estimated the relative importance of achieving each task using a weight function. We have solved our resource allocation problem in an effective manner using our optimal acyclic or approximate on-line decomposition approaches.

Evidently our decomposition algorithms can be improved in different ways. For example, computing the solution using the acyclic decomposition approach where all components are solved using the on-line decomposition approach, instead of a standard one, seems promising.

The following section proposes another decomposition technique. The present section presented a decomposition technique based on the acyclicity of the task transition graph. To further reduce the complexity of a planning problem, one can assign an agent to each resource type. The agent are coordinated using a central agent to generate a near-optimal policy.

## 6.3 An MTAMDP Approach

### 6.3.1 Introduction

In the previous section, the resources where considered as shared by the planning agent. In this section, the problem is decomposed so that a planning agent manages each specific resource. The separate policies produced by the agents, if not coordinated, are not optimal for two reasons. Firstly, some resources may have positive and negative interactions since the expectation of realizing a certain task $ta_1$ by resource $res_1$ is changed when allocating another resource $res_2$ simultaneously on task $ta_1$. Secondly, the resources have to be distributed efficiently among the tasks to accomplish. To achieve that, we have coordinated the planning agents together during the planning process through a *central* agent, so that they can produce a near-optimal policy. Each planning agents generate a policy to allocate their resources using a MDP. This method is called "Multiagent Task Associated Markov Decision Process" (MTAMDP) (Plamondon et al. (2006b); Plamondon et al. (2006a)) since the planning agents are related to each other only because they have the same tasks to accomplish. The results obtained using MTAMDP are near-optimal, while the convergence time is very small compared to a standard Multiagent Markov Decision Process (MMDP) (Boutilier et al., 1999b) approach.

#### Multi-Agent Markov Decision Processes (MMDPs)

Since resource allocation problems are known to be NP-Complete Zhang (2002), one may decompose the previous problem into multiple planning agents. To do so, Multi-Agent Markov Decision Processes (MMDPs) (Boutilier et al., 1999b) may be a very suitable modelling framework. In an MMDP the individual actions of many planning agents interact so that the effect of one agent's actions may depend on the actions taken by others. Furthermore, an MMDP takes the agents to be acting on behalf of

some individual; therefore, each has the same utility or reward function $R$. Indeed, an MMDP is like an MDP, except that $P$ now refers to the probabilities of joint actions.

In a MMDP, the joint action space is considered and $|A|$ is exponential with the number of resource types, as demonstrated in Section 3.1.1. On the other hand, if an agent plans for each resource type, as with the MTAMDP approach presented in this section, the number of actions to consider in a state is:

$$\sum_{i=1}^{nbAgents} nbChoices_i \tag{6.5}$$

Here, the number of actions is the sum of the possible actions of each resource. The number of Q-values to compute in a state, if three available resource types are available, with each ten possible actions, is $10 \times 3 = 30$. The value of $nbChoices_i$ is still, however, exponential for each planning agent, as the number of tasks augment. Indeed, a resource allocation problem is still an NP-Complete problem, even when one resource type is available (Zhang, 2002).

A *central* agent computes the Q-value of all action combinations using the new efficient approach that will be detailed in the subsequent sections. The next section describes in a more formal way MTAMDPs, which significantly reduces computational complexity associated with the high number of possible resources.

### 6.3.2 Related Work

The complexity of our problem is exponential according to the number of resources and tasks, and many approximations and heuristics have been proposed (Meuleau et al. (1998); Wu and Castanon (2004); Aberdeen et al. (2004)). However, all these authors do not consider positive and negative interactions among resources. Their approaches are consequently not very suitable to the type of problem tackled in this section, since in many real applications there are positive and negative interactions among resources. An effective approach, as considered in the current paper, is to plan for the resources separately as proposed by Wu and Castanon (2004). In this section, a different and more general approach is described, where each resource is coordinated by a *central* agent, instead of sequentially.

Russell and Zimdars (2003) propose another *central* agent coordination scheme. Their Q-Decomposition Reinforcement Learning coordination process determines the Q-values which maximize the sum for each states at each learning iteration. An important assumption of this method is that each agent should have its own independent reward function. Thus, for a resource allocation problem, there would have an agent for each task to achieve. This is different than the approach proposed in this section where there are an agent for each resource type. In their setting, their approach is optimal,

but if resource constraints are considered, the policy has to consider the entire state space. Indeed, the value of a state of a task would have a dependence on the state of the other tasks, because of resource constraints. The Q-Decomposition may be combined with the MTAMDP method to further reduce the planning time.

Since resources have local and global resource constraints on the number that be used, the problem here can be viewed as a constrained Markov decision process (Bertsekas (2005); Feinberg and Shwartz (1996); Dolgov and Durfee (2004); Altman (1999)). In this context, dynamic programming (Bertsekas (2005); Feinberg and Shwartz (1996)) or linear programming (Dolgov and Durfee, 2004) may be used to obtain a policy. Much work has been done in this field, but none of it coordinate separate agents which have positive and negative interactions among resources and produces a near-optimal policy as in this section.

Another interesting approach is that of Dolgov and Durfee (2004). They proposed a mixed integer linear programming formulation of a stochastic resource allocation problem. The planning time is greatly reduced compared to a large Multiagent Markov Decision Process (MMDP) (Boutilier et al., 1999b) on the joint state and action spaces of all agents. However, their approach supposes a static allocation of resources, while this thesis is interested in a dynamic allocation to consider contingencies. The problem is now modelled in more detail.

### 6.3.3 A Multiagent Task Associated Multiagent Process (MTAMDP) Method

An abstract schematization of the approach proposed in this section, which extends MMDPs, to solve efficiently a resource allocation problem is described in Figure 6.8 where each planning agent ($i_1$, $i_2$, and $i_3$) manages one type of resource to accomplish the tasks.

The dotted line in the Figure represents agent $i_1$ which manages resource type $res_1$ to accomplish all tasks. This way, each agent can compute a Q-value ($Q_{i_1}$, $Q_{i_2}$, $Q_{i_3}$). As a matter of fact, there are two reasons why a local value function of a planning agent needs to be adjusted, according to an action from another agent.

First of all, the resources should be divided between the tasks in the case of possible simultaneous actions. This aspect is explained with an example. Suppose there are two identical tasks ($ta_1$ and $ta_2$) in the environment, which are in a particular state.

Furthermore, there are two planning agents ($i$ and $i'$) each of which manages one unit of resource ($res_1$ and $res_2$). Suppose that using each resource is 50% likely to achieve any task (i.e. 50% likely that the task is realized, and 50% that the task is not realized). Now, suppose that both planning agents assign their respective resource to

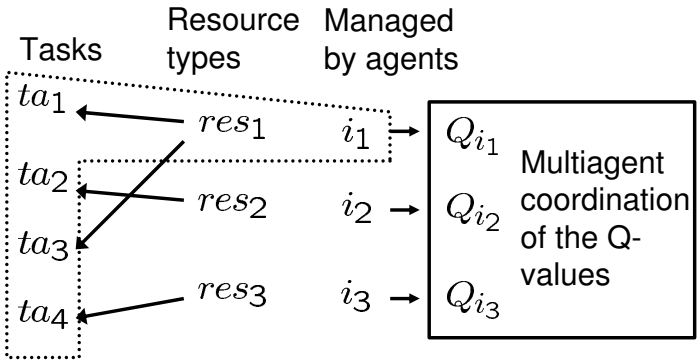



Figure 6.8: A multiagent task associated resource allocation problem (Plamondon et al., 2006b).

task $ta_1$. In these conditions, task $ta_1$ is 75% likely to be achieved[1], while task $ta_2$ still has 0% of being achieved. As a consequence, there is expected to remain $0.25+1 = 1.25$ tasks in the environment.

On the other hand, if both planning agents had coordinated and used their resources on a different task, both tasks would have been achieved at 50%. In this case, there is expected to remain only $0.5 + 0.5 = 1$ task in the environment. This simple example shows why it is better for the planning agents to divide their resources between the tasks in the case of possible simultaneous actions.

The second reason why the planning agents should coordinate is to benefit from positive interactions among resources and to alleviate the negative ones. Indeed, the expectation of realizing a certain task $ta_1$ by a unit of resource $res_1$ could be changed positively or negatively when allocating another unit of resource $res_2$ simultaneously on task $ta_1$.

A positive interaction between two resources used simultaneously produces a higher percentage to realize a task than using these two resources separately. On the other hand, a negative interaction between two resources used simultaneously produces a lower percentage to realize a task than using these two resources separately.

The general idea proposed in this section to solve efficiently the described resource allocation problem is to coordinate the different agents at each iteration of the planning algorithm. Indeed, all existing algorithms to solve an MDP are iterative, thus the approach presented here should be pretty extensible. Figure 6.9 describes this process. For example, if $n$ iterations are needed for each planning agent to converge, then $n$ coordination activities are made.

[1] $1 - 0.5^2 = 0.75$

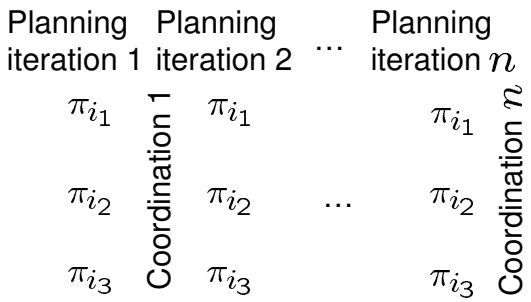



Figure 6.9: The iterative coordination process (Plamondon et al., 2006b).

### Adjusting the Q-value of a Planning Agent

The *central* agent permits reaching a near global optimum for the planning process by coordinating the different planning agents, as shown in the experiments. The two reasons why the respective policies of the planning agents should be coordinated lead to the definition of two subfunctions — ADJUST-I($a$) (interactions) and ADJUST-SA($a$) (simultaneous actions). The ADJUST-I($a$) function is defined as follows:

**Definition 6.3.1** ADJUST-I($a$)*: This function adjusts the task-Q-values of each planning agent $i$, considering a global action $a$, in a global state $s$. The adjustment is made according to the interactions between the actions of each other planning agent $i'$.*

Before detailing the calculus to obtain the adjusted Q-value according to the interactions with another action, $a_{Inter_{i_{ta}}}(a_{i_{ta}}, s|a'_{i'_{ta}}) \in a_{i_{ta}}$ is introduced as the part of action $a_{i_{ta}}$ in interaction with $a'_{i'_{ta}}$. In this context, the following definition is given:

**Definition 6.3.2** *$Inter_{a_{i_{ta}}}(a_{i_{Inter_{ta}}}(a_{i_{ta}}, s|a'_{i'_{ta}}), s|a'_{i'_{ta}})$ is the modified efficiency of action $a_{i_{Inter_{ta}}}(a_{i_{ta}}, s|a'_{i'_{ta}})$ in state $s$ knowing that action $a'i'_{ta}$ is also executed.*

For example, if $Inter_{a_{i_{ta}}}(a_{i_{Inter_{ta}}}(a_{i_{ta}}, s|a'_{i'_{ta}}), s|a'_{i'_{ta}}) = 0.7$, it means that $a_{i_{Inter_{ta}}}(a_{i_{ta}}, s|a'_{i'_{ta}})$, knowing that $a'_{i'_{ta}}$ is also used, has its efficiency at 70% of its regular one. In addition, both the ADJUST-I($a$) and ADJUST-SA($a$) functions need an upper bound on the value that a state may take in a specific iteration. The definition of this term has a great effect on the quality of the approximation obtained for the approach in this section. In particular, the upper bound is defined as follows:

**Definition 6.3.3** *$UpBound^s_{a_{i_{ta}}}$ represents the maximum value that agent $i$ may expect to obtain in state $s$, when executing action $a_{i_{ta}}$. All upper bounds are specified at each iteration of the planning algorithm.*

The heuristic used to determine the upper bound for a state $s$, and action $a_{i_{ta}}$ by agent $i$, is the highest value of a possible state transition. A possible state transition is

considered as a state for which $P_{a_{i_{ta}}}(s'|s) > 0$. This way, the upper bound overestimates the possible value of a state since it is very improbable that an action would guarantee reaching the upper bound. This upper bound provides an approximation of sufficient quality to address the problem at hand. The Algorithm 6.5 to obtain the value of ADJUST-I($a$) for a specific global $a$, in a state $s$ is now described. In the algorithm the *noOp* term signifies the no-operation action.

**Algorithm 6.5** The algorithm for considering interactions (Plamondon et al., 2006b).

1: **Function** ADJUST-I($a$)
2: **for all** $m \in Ag$ **and** $ta \in Ta$ **do**
3: **if** $a_{i_{ta}} \neq noOp$ **then**
4: $null_{a_{i_{ta}}} \leftarrow Q_{i_{ta}}(noOp_{a_{i_{ta}}}, s(Res_{i_{ta}} \setminus res(a_{i_{ta}})))$
5: $delta_{a_{i_{ta}}} \leftarrow Q_{i_{ta}}(a_{i_{ta}}, s) - null_{a_{i_{ta}}}$
6: **for all** $m' \in Ag$ different from $i$ **do**
7: **if** $a_{i'_{ta}} \neq noOp$ **then**
8: $inter \leftarrow Inter_{a_{i_{ta}}}(a_{i_{Inter_{ta}}}(a_{i_{ta}}, s|a'_{i'_{ta}}), s|a'_{i'_{ta}})$
9: **if** $inter \leqslant 1$ **then**
10: $delta_{a_{i_{ta}}} \leftarrow delta_{a_{i_{ta}}} \times inter$
11: **else**
12: $gain \leftarrow \frac{delta_{a_{i_{ta}}}}{UpBound^{s}_{a_{i_{ta}}} - null_{a_{i_{ta}}}}$
13: $nGain \leftarrow 1 - (\frac{1-gain}{inter})$
14: $delta_{a_{i_{ta}}} \leftarrow \min(delta_{a_{i_{ta}}} \times \frac{nGain}{gain}, delta_{a_{i_{ta}}} \times inter)$
15: **end if**
16: **end if**
17: **end for**
18: **end if**
19: **end for**
20: **for all** $m \in Ag$ **and** $ta \in Ta$ **do**
21: $Q_{i_{ta}}(a_{i_{ta}}, s) \leftarrow null_{a_{i_{ta}}} + delta_{a_{i_{ta}}}$
22: **end for**

Line 4 of the algorithm represents the value of an action which has an interaction of 0. The intuition is that doing nothing ($noOp_{a_{i_{ta}}}$), and subtracting the resource used by the action, has the same value as doing an action with the resource which is sure of not realizing its purpose. The results of Line 5 is the difference of utility from not considering the interactions to consider an interaction of 0. Afterwards, two cases to compute the Q-value of the current planning agent, considering an interaction with another planning agent are possible. Firstly, in Line 9, negative interactions are considered such that $inter \leqslant 1$. In this case, $delta_{a_{i_{ta}}}$ is adjusted by multiplying it with the interaction. On the other hand, if the interaction is positive (i.e. Line 11)

the adjustment is more complex. The Line 12 represents the "gain" the current action has made according to the upper bound it may reach. Then, this "gain" is reviewed considering the interaction in Line 13. The new $delta_{a_{i_{ta}}}$ is obtained by multiplying it with a fraction of the gain of the interaction and the gain without interaction. Finally, in Line 21 all task-Q-values are updated considering the new $delta_{a_{i_{ta}}}$. The update of all task-Q-values is made after all adjustment have been done, thus the reason of the summation in Line 20.

To consider simultaneous actions, the ADJUST-SA($a$) function is used, and is defined as follow:

**Definition 6.3.4** ADJUST-SA($a$)*: This function adjusts all task-Q-values of each planning agent $i$, considering a global action $a$, in a global state $s$. The adjustment is made according to the simultaneous actions of all other planning agents $i'$.*

Algorithm 6.6 details the ADJUST-SA($a$) function. The first part of the algorithm (i.e. Lines 4 to 12) finds the highest upper bound, considering all agents. The second part (i.e. Lines 13 to 19) calculates two terms. Firstly, *sum* represents what the agents expects to gain by planning for their action, by planning independently. Then, *val* computes the maximum value that all agents may have, considering that they are planning on the same task. When these two terms are computed, in Line 22 the gain of value of planning for this action is multiplied by the fraction $val/sum$. In this line, the new Q-value is also affected to this "adjusted" gain. Like in the previous algorithm, the update of all task-Q-values is made after all adjustment have been done.

Now, the *central* agent knows how to compute the adjusted Q-value of each planning agent in a state, given the action of the other planning agents. The other function the *central* agent uses in its coordination process is the GLOBAL-VALUE() one.

### Determining the Value of a Global Action

To determine the action to execute in a state, the *central* agent has to calculate a global Q-value, considering each planning agent Q-value. This is done in a precise manner by considering the task-Q-values. Before introducing the algorithm, we recall that a planning agent for each resource type, and a *mNoOp* planning agent for the *noOp* (no operation) action are considered. The *noOp* action has to be considered since this action may modify the probability to achieve certain tasks in a state.

Algorithm 6.7 determines a precise value of the adjusted task-Q-value of many planning agents in a state. The central part of this algorithm is in Line 6 where the value of a task is computed, according to all agents. Here, the maximum between the Q-value of the current agent $i$, and the gain that the current agent may offer is taken as the value *val*. In this case, the Q-value of a planning agent $i$ is subtracted by the

**Algorithm 6.6** The algorithm for considering simultaneous actions (Plamondon et al., 2006b).

1: **Function** ADJUST-SA($a$)
2: **for all** $ta \in Ta$ **do**
3: $bound \leftarrow R(s_{ta}|s_{ta} = \neg goal)$
4: **for all** $m \in Ag$ **do**
5: **if** $a_{i_{ta}} \neq noOp$ **then**
6: $null_{a_{i_{ta}}} \leftarrow Q_{i_{ta}}(noOp_{i_{ta}}, s(Res_{i_{ta}} \setminus res(a_{i_{ta}})))$
7: **if** $UpBound^{s}_{a_{i_{ta}}} > bound$ **then**
8: $bound \leftarrow UpBound^{s}_{a_{i_{ta}}}$
9: $noOpG \leftarrow null_{a_{i_{ta}}}$
10: **end if**
11: **end if**
12: **end for**
13: **for all** $m \in Ag$ **do**
14: **if** $a_{i_{ta}} \neq noOp$ **then**
15: $delta_{a_{i_{ta}}} \leftarrow Q_{i_{ta}}(a_{i_{ta}}, s) - null_{a_{i_{ta}}}$
16: $sum_{ta} \leftarrow sum_{ta} + delta_{a_{i_{ta}}}$
17: $val_{ta} \leftarrow val_{ta} + (((bound - noOpG) - val_{ta}) \times (\frac{delta_{a_{i_{ta}}}}{UpBound^{s}_{a_{i_{ta}}} - null_{a_{i_{ta}a}}}))$
18: **end if**
19: **end for**
20: **for all** $m \in Ag$ **do**
21: **if** $a_{i_{ta}} \neq noOp$ **then**
22: $Q_{i_{ta}}(a_{i_{ta}}, s) \leftarrow null_{a_{i_{ta}}} + (delta_{a_{i_{ta}}} \times \frac{val_{ta}}{sum_{ta}})$
23: **end if**
24: **end for**
25: **end for**

$mNoOp$ planning agent, because each agent considers also the $noOp$ action. The last function that the *central* agent needs to coordinate the planning agent in a near-optimal manner at each iteration is described; the adaptation of the value iteration (Bellman, 1957) algorithm for MTAMDPs, is presented in the following section.

### Value Iteration for MTAMDPs

The value iteration MTAMDP algorithm is presented in Algorithm 6.8. In lines 6 to 11 of this algorithm, a Q-value is computed for all task-state-action tuples for each planning agent. The agents are limited by the local and global resource constraints while planning their respective policies. Afterwards, in lines 15 and 16, the *central* agent

**Algorithm 6.7** The algorithm for Computing the Q-value of a state (Plamondon et al., 2006b).

1: **Function** GLOBAL-VALUE()
2: $Q(a, s) \leftarrow 0$
3: **for all** $ta \in Ta$ **do**
4: $val \leftarrow Q_{ta}^{mNoOp}(a_{ta}^{mNoOp}, s)$
5: **for all** $m \in Ag$ **do**
6: $val \leftarrow \max(Q_{i_{ta}}(a_{i_{ta}}, s), val + ((R(s_{ta}|s_{ta} = goal) - val) \times (Q_{i_{ta}}(a_{i_{ta}}, s) - Q_{ta}^{mNoOp}(a_{ta}^{mNoOp}))))$
7: **end for**
8: $Q(a, s) \leftarrow Q(a, s) + val$
9: **end for**

adjusts the value of all action combinations, in all possible states using the ADJUST-I($a$) and ADJUST-SA($a$) functions (i.e. Algorithm 6.5 and 6.6). When the adjusted value of each action is determined, the global value $V'(s)$ is computed in Line 17. If this global Q-value is the maximum one at present, the value of each planning agent is assigned to the adjusted Q-value (i.e. $V_i'(s_i) \leftarrow Q_i(a_i, s_i)$ in Line 20). The new value of the state is also assigned to the global value obtained by GLOBAL-VALUE() (i.e. $V'(s) \leftarrow Q(a, s)$ in Line 22). When the global value function has converged, the agents use their own policy for execution. All the performed experiments, as presented in the next section, resulted in a convergence. We have not a formal theorem to prove the convergence of MTAMDPs, nor its near-optimality. These proofs are for future work.

#### Merging the Acyclic Decomposition and MTAMDP Approaches (MTAMDP Decomposition)

The merging of the acyclic decomposition of Section 6.2.3 and the MTAMDP approaches is pretty straightforward . Indeed the line $V_c \leftarrow$MDP-ALGO($c, \epsilon$) in Algorithm 6.3 is now $V_c \leftarrow$MTAMDP-VI($c$, $\epsilon$) (Plamondon et al., 2006a). The fact that only this simple change is needed demonstrates the flexibility and extensibility of both these approaches.

### 6.3.4 Experiments and Discussion

Modelling a stochastic resource allocation problem using the MTAMDP decomposition approach allows reducing the number of actions to consider in a given state. In particular, the difference in complexity between MMDPs and MTAMDPs resides in the reduction of the computational complexity from using the complex Bellman

**Algorithm 6.8** The value iteration for MTAMDPs algorithm (Plamondon et al., 2006b).

**Function** MTAMDP-VI($S$, $\epsilon$)
**returns** a value function $V^{Ag}$
{Planning agents part of the algorithm}
**repeat**
  $V \leftarrow V'$
  $\delta \leftarrow 0$
  **for all** $m \in Ag$ **do**
    $V_i \leftarrow V_i'$
    **for all** $s_i \in S_i$ **and** $a_i \in A_i(s)$ **do**
      $Q_i(a_i, s_i) \leftarrow R(s_i) + \sum_{s_i' \in S_i} P_{a_i}(s_i'|s_i) V_i(s_i'(Res_i \setminus res(a_i)))$
    **end for**
  **end for**
  {*Central* agent part of the algorithm}
  **for all** $s \in S$ **do**
    $V'(s) \leftarrow R(\neg s)$
    **for all** $a \in A(s)$ **do**
      ADJUST-I($a$)
      ADJUST-SA($a$)
      $Q(a, s) \leftarrow$ GLOBAL-VALUE()
      **if** $Q(a, s) > V'(s)$ **then**
        **for all** $m \in Ag$ **do**
          $V_i'(s_i) \leftarrow Q_i(a_i, s_i)$
        **end for**
        $V'(s) \leftarrow Q(a, s)$
      **end if**
    **end for**
    **if** $|V'(s) - V(s)| > \delta$ **then**
      $\delta \leftarrow |V'(s) - V(s)|$
    **end if**
  **end for**
**until** $\delta < \epsilon$
**return** $V^{Ag}$

equation by MMDPs, in contrast to using the ADJUST-I($a$), ADJUST-SA($a$), and GLOBAL-VALUE() functions when computing the value of each action combination by MTAMDPs.

The domain of the experiments is as described in Section 4.7. Furthermore, the

different resources have their efficiency modified when used in conjunction on a same task, thus producing positive and negative interactions among resources. Also, a non-critical task may create a critical task, but not vice versa, thus the task transition is acyclic.

We have compared four different approaches (Plamondon et al., 2006a). The first one is the MMDP approach as described briefly in Section 6.3.3. MMDP is computed as a traditional "flat" MDP on the joint action and state spaces of all agents. The second approach is the MTAMDP as described in Algorithm 6.8. The third one is the "*one step* MTAMDP" where the adjustment of the Q-values is made only at the last iteration for the planning agents. Thus, when each planning agent has converged, the Q-values of all agents are adjusted, and used for execution. The fourth approach is the "*no coordination*" where each planning agent plans resources completely independently of each other. We have compared these four approaches in both the standard (no acyclic decomposition), and acyclic decomposition mode, to efficiently consider the creation of tasks by other tasks.

We compare the MTAMDP approach with an MMDP approach in Figure 6.10, where each agent manages a distinct resource type. The acyclic decomposition algorithm further reduces the planning time for the four different approaches as presented in Figure 6.11. The results are very encouraging. For instance, it takes 51.64 seconds to plan for an acyclic decomposition MTAMDP approach with five agents. The acyclic decomposition MTAMDP approach solves the same type of problem in an average of 0.72 seconds.

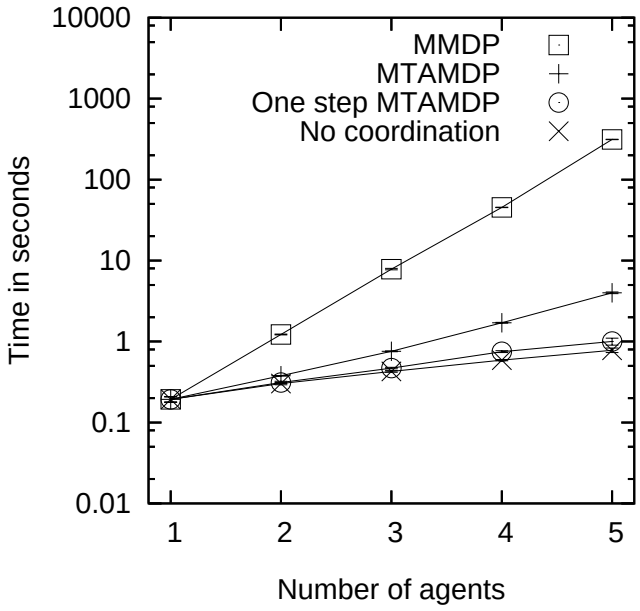



Figure 6.10: Planning time using no acyclic decomposition (Plamondon et al., 2006a).

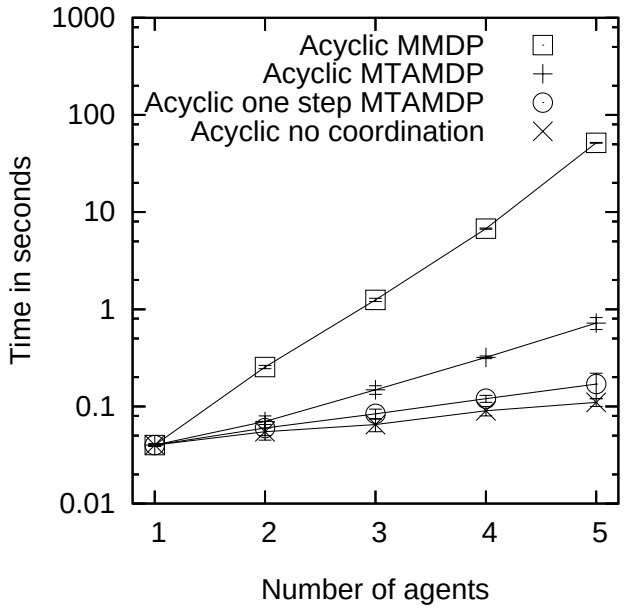



Figure 6.11: Planning time using the acyclic decomposition (Plamondon et al., 2006a).

Table 6.4 details how far the expected value of the MTAMDP, *one step* MTAMDP, and *no coordination* approaches are from an optimal MMDP approach. With one agent, all approaches are optimal since no coordination is needed. This result suggests that the GLOBAL-VALUE() function is optimal. All the tests performed with two agents resulted in an optimal policy for the MTAMDP approach. This result suggests that the GLOBAL-VALUE() function is optimal, and a formal theorem to proof that is for future work. One can also observe that when the agents do not coordinate, the resulting policy is far from the optimal, which is not the case for the MTAMDP coordination approach. The one step MTAMDP could be a viable approach in certain critical situations, since the solution is produced much faster than the MTAMDP approach while providing a near-optimal policy.

Table 6.4: The percentage of the optimal obtained with the different approaches (Plamondon et al., 2006a).

| | MTAMDP | One step MTAMDP | No Coordination |
|---|---|---|---|
| 1 agent | 100% | 100% | 100% |
| 2 agents | 100% | 99.89% | 97.48% |
| 3 agents | 99.84% | 99.63% | 94.20% |
| 4 agents | 99.79% | 99.55% | 91.63% |
| 5 agents | 99.67% | 99.41% | 89.37% |

A new approach has been proposed to tackle the planning problem for resource allo-

cation in a stochastic environment. The Multiagent Task Associated Markov Decision Process (MTAMDP) (Plamondon et al. (2006b); Plamondon et al. (2006a)) framework has been introduced to reduce the computational leverage induced by the high number of possible resources. The experiments has shown that the MTAMDP provides a potential solution to solve efficiently real-time stochastic resource allocation problems with positive and negative interactions.

A way to improve the proposed MTAMDP consists of providing a more precise upper bound on the value that a Q-value may have. Currently, the approach simply considers the maximum value of the possible states transitions. This way, the upper bound overestimates the possible value of a state since it is very improbable that an action would guarantee reaching the upper bound. Other heuristics need to be explored to define the upper bound.

The Q-Decomposition approach proposed by Russell and Zimdars (2003) is approximative for the resource allocation problem considered here. However, it may be combined with the MTAMDP method to further reduce the planning time. Indeed, the Q-Decomposition would decompose the problem in tasks, and the MTAMDP method decomposes the problem in resources. Thus, this would permit two degrees of decomposition.

Another way to improve MTAMDPs may be to coordinate the agents using a Partial Global Planning (PGP) (Decker and Lesser, 1992) approach instead of a *central* agent. The PGP approach solves the bottleneck effect induced by the *central* agent. Furthermore, the coordination will be more precise, as only interacting agents will coordinate with each other. Other promising future work is to extend the MTAMDP approach with that of Singh and Cohn (1998) to use an upper bound and a lower bound on the value of each state. The Singh and Cohn approach needs some modifications to be used for the resource allocation problem considered here, as positive and negative interactions were considered here. The idea is to solve the problem using the MTAMDP approach, but with a upper and lower bounds on the value of each state.

The next chapter details the implementation of LRTDP in a very realistic professional simulator.

# Chapter 7

# Implementation in Surface Air Defence Model (SADM)

## 7.1 SADM

This chapter explores a more realistic implementation of the problem described in Section 4.3.2. We solve the problem using heuristic search, where in particular, Labeled Real-Time Dynamic Programming (LRTDP) (Bonet and Geffner, 2003b) is applied.

In BAE System's Surface Air Defence Model (SADM) (SADM), the time for weapon to intercept a threat depends on the range, the type and the speed of the threat. In addition, weapons cannot fire at threats freely. Some constraints apply depending on their incoming azimuth and their distance from the own platform. In our example, we consider that the own platform is equipped with two Separate & Track Illumination Radar (STIR), two Ship-Air Missiles (SAM) launchers, a Gun and a Close-In Weapon System (CIWS) for hardkills and with four chaff launchers and one jammer for softkills. To ease example understanding all threats are assume to be the same type but with different starting range and speed[1]. The table 7.1 describes how weapons are constrained and what are their probability of success.

In our application, weapons are constrained during an episode: There exists unary constraints that specify which threats are reachable regarding their distance from the own platform and the range of weapon, and binary constraints that bind firing weapons to STIRs depending on threats STIRs can "see". Few constraint of resource limitations already exists.

On the other hand, softkill modelization is more complex. In our model, chaffs can be launched following four fixed directions given in Figure 7.1 and using two mode, seduction or distraction. Seduction must be used when a threat already locked the own

[1]Threats speed is assumed to remain constant during an engagement.

**Hardkill Constraints**

$C_1$: A SAM must be guided by a STIR from fire time to interception time,

$C_2$: A Gun must use a STIR at fire time,

$C_3$: Two STIRs cannot target the same threat.

**Hardkill Blind Zones**

| Base State | 0 to 360° | 1 STIR, 1 Gun, 1 CIWS, 2 SAMs |
|---|---|---|
| Sector | Angles | Difference from Base state |
| A | 345 to 15° | No CIWS |
| B | 15 to 60° | No difference – Base state |
| C | 60 to 120° | An additional STIR |
| D | 120 to 145° | No difference – Base state |
| E | 145 to 215° | No Gun |
| F | 215 to 240° | No difference – Base state |
| G | 240 to 300° | An additional STIR |
| H | 300 to 345° | No difference – Base state |

| Weapon | Range | Probability of success |
|---|---|---|
| SAM | From 2.2 to 20km | 95% |
| Gun | From 1.5 to 5km | 50% |
| CIWS | From 0.2 to 2km | 10% |

Table 7.1: Examples of problem's constraints.

platform so as to propose another target to it, and then using jammer to deviate it on the seduction-mode chaff. Distraction must be used when a threat has not locked yet so as to propose another target before it locks. Chaffs effectiveness is unknown *a priori* and probably depends on the azimuth of threats. Furthermore, the jammer may be coordinated with chaff to increase their effectiveness. However, if a chaff is addressing a threat or a group of threats, it may affect the effectiveness of hardkills since a chaff deceive threats as it disturb STIR's illumination and targeting. Thus, we first learn what are softkills effectiveness alone and then try to coordinate them with hardkills. We now introduce the reflex planner. Later in this chapter, we compare an LRTDP implementation with this reflex planner.

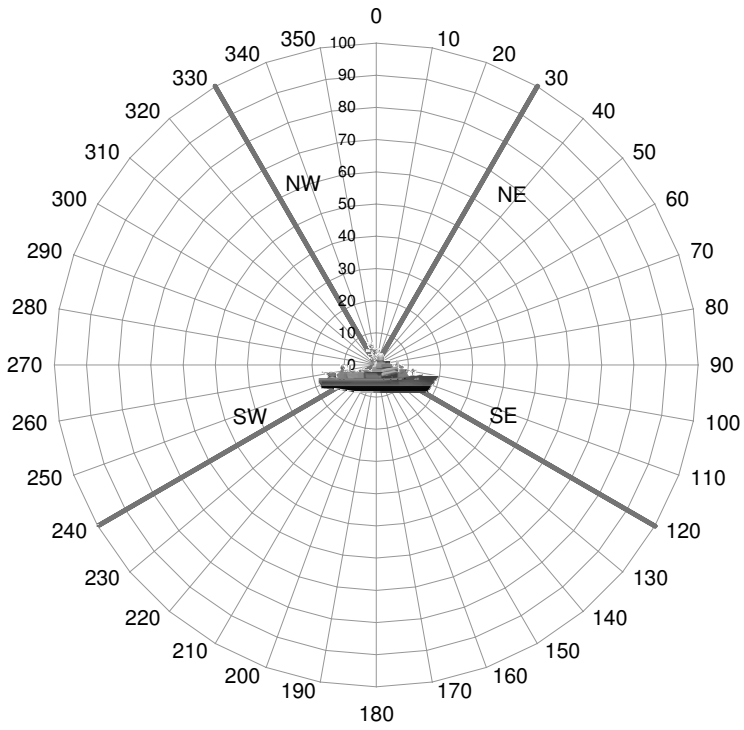



Figure 7.1: Directions where chaff can be launched. NE = north east, NW = north west, SE = south east, SW = south west.

## 7.2 Reflex Planner

The following sections describe the reflex planner (Besse et al., 2007) which has been implemented in SADM.

### 7.2.1 Softkill Basic Behavior

As chaffs effectiveness is unknown *a priori*, their probability distributions of deceiving a threat is evaluated with Monte-Carlo simulations and estimations as proposed in Sutton and Barto's Sutton and Barto (1998) book. These distributions models what will be the softkill agent behavior. As a consequence, the softkill agent will have to choose an action to do among sixteen possible action (4 directions × 2 modes × Jammer or not) each time it should engage a threat. We first estimate the effectiveness of one chaff against one threat and obtain probability distributions of deceiving a threat depending on action chosen and azimuth of the threat.

For instance, the effectiveness of a chaff launched in seduction mode is shown in Figure 7.2 and the same located chaff launched in coordination with a jammer is also

shown in Figure 7.2. This figure shows the percentage of *sofkilled* threats (i.e. threats countered with softkill weapons) depending on their incoming azimuth. Results are obtained by simulating apparition of threats uniformly around the ship and evaluating the percentage of deceiving each chaff have depending on azimuth and jammer utilization.

The loss of efficiency by using the jammer from $320^o$ to $360^o$ is due to the fact that jammer may attract the threat on the own platform instead of deceiving it if it is used too longer. This fact must also be taken into account in the coordinating agent, which merges the hardkill and softkill plans.

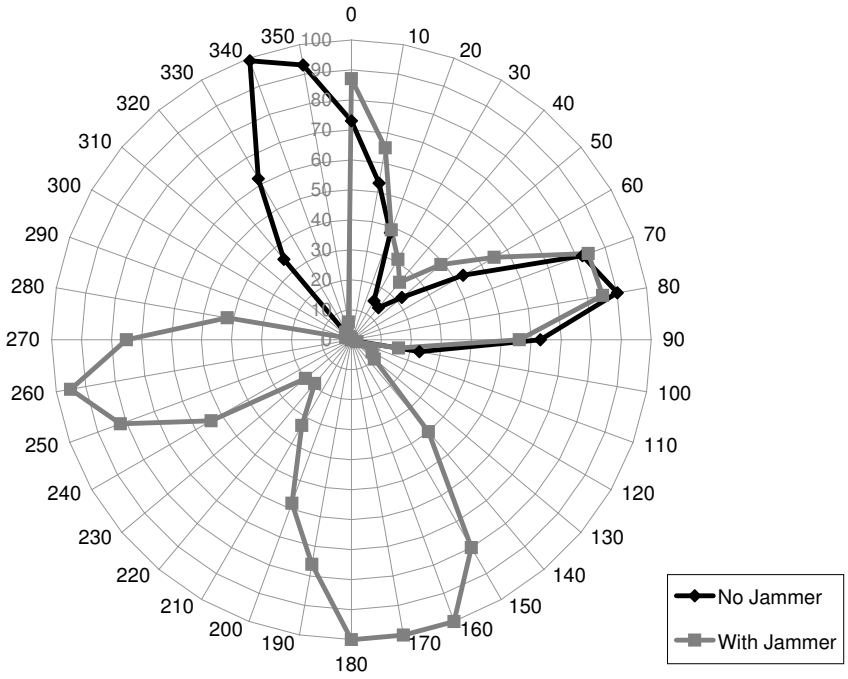



Figure 7.2: North-East ($30^o$ ) Seduction Chaff. Light gray curve shows the coordination with the Jammer, the black line shows no coordination with the Jammer

### 7.2.2 Developed Policies

Once we obtained the probability distributions, some theoretical policies could be developed. In fact, these policies are the product of a conjunction of measurements following a given criteria. For instance, the policy given by Figure 7.3 ensures the best chaff to launch to counter a threat and what is the gain if we synchronize it with the jammer. Samples of the chaff and the mode to choose and its probability of success to counter the threat is given in table 7.2.

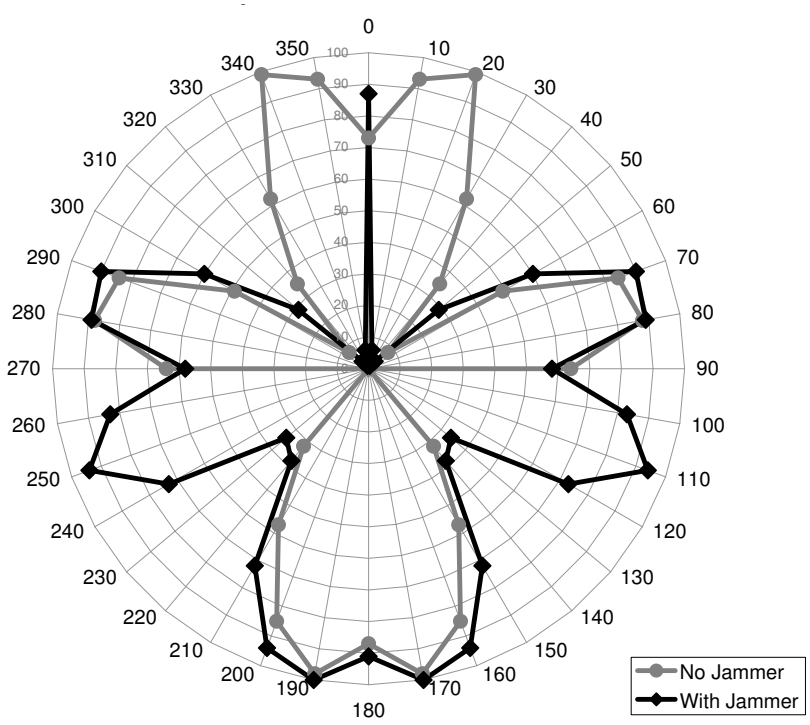



Figure 7.3: Policy % effectiveness based on results of learning. Light gray curve shows the chaff-only policy, the black line shows improvement made by synchronizing the Jammer with the chaff.

| | % No Jammer | % Jammer | Action |
|---|---|---|---|
| 0 | 73 | 87 | NE Seduction |
| 10 | 93 | 6 | NW Seduction |
| 20 | 99 | 1 | NW Seduction |
| ... | ... | ... | ... |
| 120 | 0 | 73 | SW Seduction |
| 130 | 0 | 34 | SW Seduction |
| 140 | 32 | 38 | SE Seduction |
| 150 | 57 | 72 | SE Seduction |
| 160 | 85 | 94 | SE Seduction |
| ... | ... | ... | ... |
| 220 | 32 | 38 | SW Seduction |
| 230 | 0 | 34 | SE Seduction |
| 240 | 0 | 73 | SE Seduction |
| ... | ... | ... | ... |
| 340 | 99 | 1 | NE Seduction |
| 350 | 93 | 6 | NE Seduction |

Table 7.2: Policy % effectiveness based on results of learning. Chaff to launch is indicated in the *Action* column.

### 7.2.3 Policies Evaluation

Allowing the ship to maneuver could, in fact, improve softkill defense in some cases but could also decrease it in other cases. Thus, we decided to improve ship's survivability by allocating blind-zone incoming threats to a reflex hardkill agent. This aspect of coordination will be explained in Section 7.3.3.

Results of such implementation show that chaff are symmetrically effective in first approximation but there are some variance due to uncertainty and complexity of models that simulator employs. Thus, an empirical policy for a pure reflex softkill agent is shown in Figure 7.4. This policy will be used by the reflex and LRTDP planners.

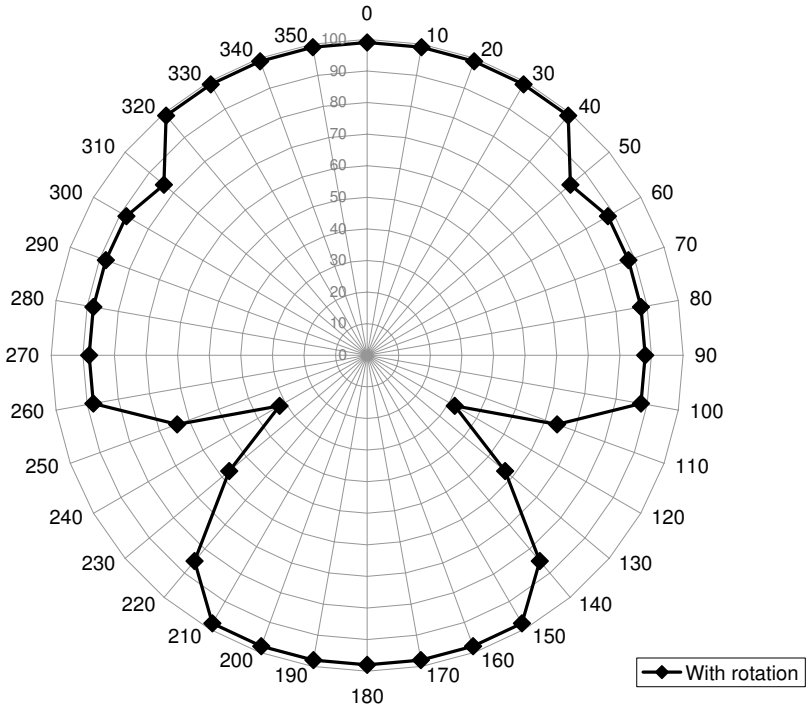



Figure 7.4: Policy % effectiveness based on empirical tests.

## 7.3 Softkill & Hardkill Planning

As stated before, we have two agents, one for the softkill system and the other for the hardkill system. When they face one or several threats, these two specific agents plan the use of weapon resources of the ship for countering the threats. Planning weapon resources in this context means allocating and scheduling the deployment of the ship's weapon resources against threats with a precise order on the intervention time.

### 7.3.1 Softkill Reflex Planning Agent

Generally, reflex planning uses very low-level reasoning techniques for a simple response to a situation to give a very short response time. This is very important in our context because defending ship brings a very hard and usually very short time constraint.

Based on policies described in previous section, the softkill agent first wait until one threat begin to search after the ship (Line 3 of algorithm 7.1). Then, the softkill agent uses known threats to regroup them (Line 6) in order to turn the ship to minimize number of threats in blind zones (Line 10). Finally, it apply policy given by methodology explained above.

**Algorithm 7.1** The softkill agent algorithm.

1: **Inputs:** *Threats*: Threats list;
2: *Policy*: Policy given in section 7.2.3;
3: **Wait until** threats going to lock.
4: {Threats pre-treatment:}
5: $Threats \leftarrow$ EvaluateAndOrder($Threats$)
6: $ThreatGroups \leftarrow$ Group($Threats$)
7: $ThreatGroups \leftarrow$ Evaluate($ThreatGroups$)
8: {Maneuvers:}
9: {Choose direction which maximize softkill effectiveness:}
10: $newHeading \leftarrow$ BestDirection($ThreatGroups$)
11: SetShipSpeed($maxSpeed\ knots$)
12: SetShipHeading($newHeading$)
13: wait for $currentHeading = newHeading$
14: SetShipSpeed($baseSpeed$)
15: {Launch chaff according to given policy}
16: LaunchChaff($Policy$)

### 7.3.2 Hardkill Reflex Planning Agent

To construct a reflex plan, the hardkill agent maintains a list of threats coming on the ship. This list is sorted according to some threat evaluation (i. e., the list is sorted from the most to the least dangerous threat). Then, it applies some predefined rules for allocating the resources. These predefined rules are given by algorithm 7.2. The algorithm first evaluates the threat rankings in Line 4, based on the closest point of approach and time of flight to the own platform. Then we assign a SAM and the Gun

to the most dangerous threat and only the SAM to the second most dangerous threat.

**Algorithm 7.2** The hardkill agent algorithm.

1: **Inputs:** $Threats$: Threats list;
2: **Wait until** threats into range.
3: {Threats pre-treatment:}
4: $Threats \leftarrow$ Evaluate($Threats$)
5: $1^{st}Threat \leftarrow$ First($Threats$)
6: $2^{nd}Threat \leftarrow$ Second($Threats$)
7: {allocating a SAM and a gun to the most dangerous threat}
8: AssignSAM($1^{st}Threat$)
9: AssignGUN($1^{st}Threat$)
10: {allocating a SAM to the second most dangerous threat}
11: AssignSAM($2^{nd}Threat$)
12: {allocating the CIWS to all threats that enter into the CIWS's range.}
13: AssignCIWS($AllThreatsInRange$)

Though these rules are simple, they allow using all available resources in an efficient way. Unfortunately, the available resources are only allocated to the two most dangerous threats, and all others in the list (if any) are not considered in the reflex plan. In the case where a kill assessment indicates that a hostile threat has been destroyed, the resources that have been allocated to this threat become available for the next most dangerous threat in the list.

### 7.3.3 Hardkill & Softkill Coordination

There are many ways to coordinate the hardkill and softkill agents. For instance, Blodgett et al. (2002) used a Central Coordinator to merge plans after receiving them from each agent. If there are some negative interactions between the planned actions, it will modify the plans to eliminate these negative interactions, or if not possible, it will try to reduce their effects.

Another option is to use a direct method where agents communicate with each other and try to coordinate their actions. In this case, communications can be used for commitments and convention as suggested by Jennings (1994), and they can be used for synchronizing plans and conflict solving.

A third method might be a kind of whiteboard (a common data space) in which the hardkill and softkill agents will construct a coordinated plan by some successive refinements. In this case, the coordination will be implicit because they will work on the same plan. Similar to this is the mediator, which in fact plays the role of a Central

Coordinator with the possibility of communication and negotiation with softkill and hardkill agents on synchronizing plans and conflicts resolution. In fact, many different communications ways could be used with their own advantages and drawbacks.

However, methods which use communication and negotiation are time consuming and therefore could decrease the quality of the produced plan. For this reason, we investigated for a centralized coordinator that does not use communication between agents. In fact, we choose the softkill agent to coordinate actions due to its large perception and the fact that we want to prioritize it higher than the hardkill one. As a result the softkill agent will avoid the hardkill agent to engage the same threats as the softkill one does. To do so, the softkill agent just decreases their priority in hardkill agent's priority list. The corresponding algorithm is given by algorithm 7.3. In Lines 6 to 12, The algorithm finds the group of threat with the best seduction probability. Then, in Lines 13 to 17, the priority of each threats is modified according to the softkill seduction probability. Finally, in Lines 19 and 20, the softkill and hardkill plan is generated. In particular, the hardkill agent will consider threats engaged by the softkill one once it has destroyed all threats with a better priority. We now describe a deliberative hardkill planner based on heuristic search, introduced in Chapter 5.

**Algorithm 7.3** The hardkill-softkill coordination algorithm.

1: **Inputs:** $Threats$: Threats list;
2: **Wait until** threats into range.
3: {Threats pre-treatment:}
4: $Threats \leftarrow$ Evaluate($Threats$)
5: {in a same manner as hardkill agent does}
6: **for all** group $g \in Threats$ **do**
7: {get the best threat group to engage}
8: $P_g \leftarrow$ SeductionGroupProba($g$, $Policy$)
9: **if** $P_g >$ SeductionGroupProba($BestGroup$, $Policy$) **then**
10: $BestGroup \leftarrow g$
11: **end if**
12: **end for**
13: **for all** Threat $i \in BestGroup$ **do**
14: {decrease their priority according to the probability to deceive it}
15: $P_i \leftarrow$ SeductionProba($Threat$, $Policy$)
16: $Priority_i \leftarrow Priority_i \times P_i$
17: **end for**
18: {Plan Softkill and Hardkill}
19: Softkill planning on $BestGroup$
20: Hardkill planning on $Threats$ updated

## 7.4 LRTDP Approach Implemented in (SADM)

Labelled Real-Time Dynamic Programming (LRTDP) (Bonet and Geffner, 2003b), which is detailed in Section 5.1.2 has been implemented in SADM. To implement our LRTDP algorithm, we modify the MDP model, defined in Section 4.7.2, as follows:

- A state $s \in S$ is a tuple $\langle t_{start}, t_{end}, Ta, alloc \rangle$. In particular, $t_{start}$ is the start time of the state; $t_{end}$ is the end time of the state; $alloc$ is a set $alloc \in A(s)$ of allocations which are already in execution at time $t_{start}$. In this implementation, a goal state is reached when no threats remain in the environment.
- An allocation consists of using a resource from a start time $t_{start}$ to an end time $t_{end}$, with the intention of countering at least one threat. An action may contains multiple allocation. The possible actions are limited by the amount that may be used on a task at a particular time. Furthermore, $t_{end}$ is an estimated upper bound of the real ending time since the action durations are uncertain.
- Each threat has the same weight, since each threat cause the same damage to the ship.
- A reward of $1 \times w_{ta}$ is given when the state of a task ($s_{ta}$) is in an achieved state, and 0 in all other cases.
- The discount factor is set to 1.

The implementation of LRTDP in SADM has been made for hardkill weapons. The algorithm is launched each time a new threat is perceived in the environment. The softkill weapons are considered as an input to LRTDP. We chose to separate the softkill and hardkill planners for LRTDP because a softkill action is usually efficient in only one range for each threat; reengagements are not effective for the softkill weapons. Since the reengagements are not effective, a dynamic programming algorithm as LRTDP is useless for softkill weapons. As previously explained in the context of coordination, the softkill reflex planner is first performed on the current threats and consequently it is considered as an input to the LRTDP hardkill planner agent. Indeed, the softkill action simply diminishes the reward we can obtain to counter a threat in respect with the kill probability of the softkill action. For example, if the reward to counter a threat is 1 and the softkill action has an 83% probability to counter this threat, the LRTDP planner considers now the reward of this threat as 0.17[2].

As in any heuristic search algorithm, a heuristic is needed for newly generated states. We used Equation 5.1 to generate this upper bound heuristic at the start of the planning process.

[2]reward - softkill action probability = new reward considered by the harkill agent (1 - 0.83 = 0.17).

When the time is introduced into our problem, matching task states to the global state is not obvious (Besse et al., 2006). Indeed, the start time and end time of a global state are generally different of the start time and end time of the specific state of each task. This is caused by the fact that the end time of a state $s$ is obtained according to the time of the first action to end when considering all tasks. On the other hand, the end time of a task state $s_{ta}$ is obtained with the first action to end, when considering the current task $ta$ only. As described by Besse et al. (2006), to match correctly a task state $s_{ta}$ within a global state $s$, we find a task state for which:

$$t_{start_{ta}} \leqslant t_{start} < t_{end_{ta}} \tag{7.1}$$

where $t_{start_{ta}}$ and $t_{end_{ta}}$ are, respectively the starting and ending time of $s_{ta}$. Also, $s_{ta}$ has to match the other characteristics of the task $ta$ in the global state $s$.

Here, the best matching state for a task is found. It is a matching state since the start time of $s_{ta}$ is less or equal than the start time of the global state $s$. Indeed, the possible actions in state $s_{ta}$ are equal or greater than the possible actions in state $s$ for task $ta$. Also, it is the *best* matching state because one and only one state for a task can satisfy Equation 7.1 and it is the state which has the start time the nearest possible of $t_{start}$, but for which $t_{start_{ta}}$ is not greater than $t_{start}$.

### 7.4.1 Issues and Limitations

LRTDP is guaranteed to return the optimal policy, given the current model and sufficient planning time. Since the algorithm is optimal, we have to consider all reachable states from the initial state, following the optimal policy. The number of possible reachable states is highly related to the number of possible hardkill engagements that can be made on each threat. For example, if we are in a state $s_0$ with five threats that can be reengaged four times each with both STIRs; we can do three actions on each threat that are:

- Engage the threat with a SAM using STIR A.
- Engage the threat with a SAM using STIR B.
- Do nothing on this threat.

In this case, there are $3 \times 5 = 15$ possible single actions that can be executed in $s_0$. However, to obtain all possible joint actions $A(s_0)$, we have to determine all possible combinations of these 15 possible actions. Since we have two STIRs, there are at most $15^2$ possible actions to execute in $s_0$. Nevertheless, there are actions that are not permitted, for example, assigning a threat with both STIRs. However, these actions are a little subset of $A(s_0)$.

All the actions in the set transit into different states, which can also have the same number of possible actions. In this example, there are $A(s_0)^d = 15^{24} = 2\ 562\ 890\ 625$ possible states, where $d$ is the length of the horizon. If $d$ is only two per threats, the number of states is only 50 625 which is significantly lower. As our approach suffers greatly in complexity if the number of reengagements for each threat is high, the characteristics of the threats have to permit a limited number of possible reengagements.

## 7.5 SADM's Configuration

In SADM, we have used a scenario where the threats appears at any degree from the ship. All threats appear in the first 10 seconds of the scenario at a range varying from 25 km to 35 km. The speed of the threats is between 600 m/s and 650 m/s to permit only 2 to 3 possible reengagements per threats.

## 7.6 Experiments for SADM

The LRTDP approach can be compared with the reflex planner. In particular, the only difference is for the hardkill weapons, which are scheduled first with simple rules for the reflex planner and then it uses a LRTDP algorithm. We will be able to compare the planning time and the survivability of the ship for both planners.

The 95% error ($Error$) probability of the ship survivability ($PS$) form Mitchell (1997) is:

$$Error = \pm\sqrt{(PS * (1 - PS))/nbTrials} \tag{7.2}$$

, where $nbTrials$ is the number of Monte-Carlo trials. For our tests, we did 500 Monte-Carlo simulations per number of threats for each approach.

Figures 7.5 and 7.6 detail the platform survival rate for the LRTDP, the reflex (Besse et al., 2007) and Tabu search (Blodgett et al., 2003) approaches for hardkill and hardkill + softkill planners. In this context, 500 Monte-Carlo trials were made for each result's point. In general, the platform survival rate is higher for LRTDP than for the reflex approach for a number of scenarios varying between 1 to 4 threats. On the other hand, the survivability is higher for the reflex planner than for LRTDP for scenarios of 5 to 8 threats. The Tabu search approach is generally less effective than the two other approaches.

Table 7.3 details the average waiting time in seconds for the LRTDP and Tabu search approaches for hardkill weapons. To measure this, we sum whenever an allocation is finished, the subtraction of the kill time that was forecasted with the actual kill time. These waiting times are due to the uncertainty of the allocation durations. The length

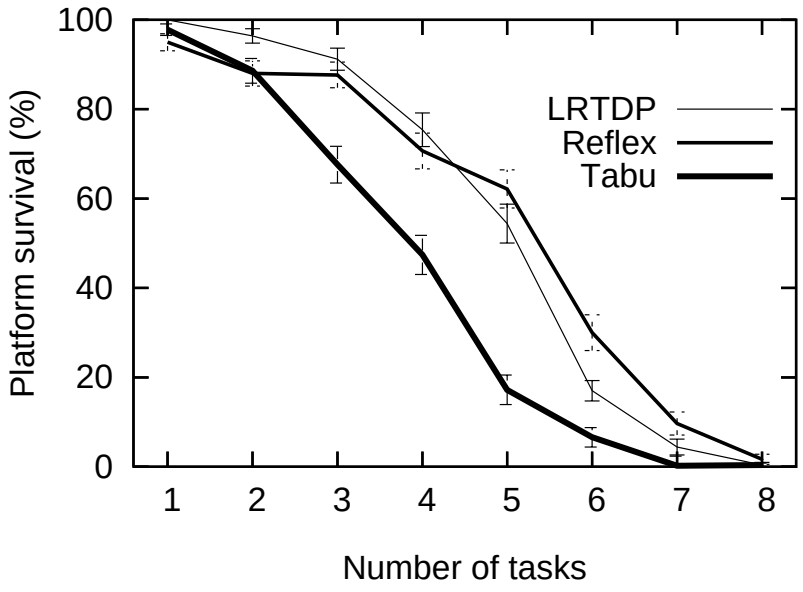



Figure 7.5: Own platform survival for hardkill with the LRTDP, reflex and Tabu search approaches.

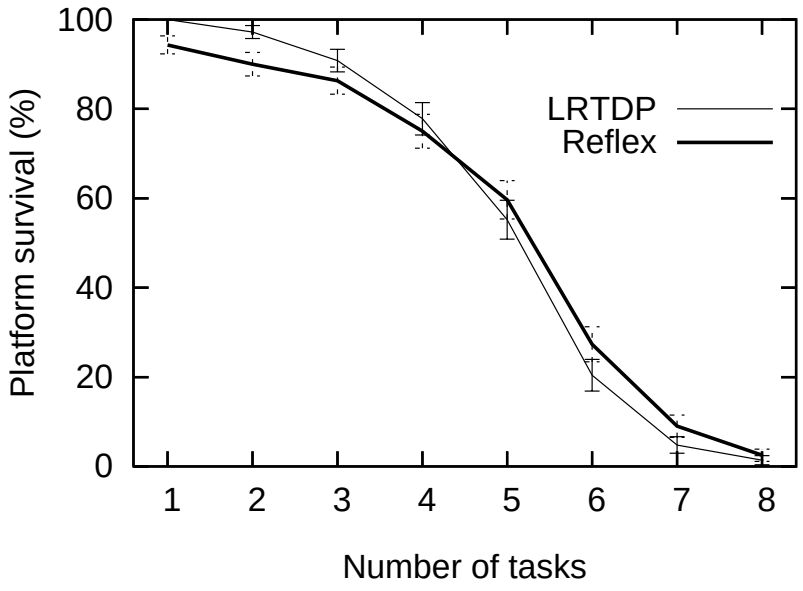



Figure 7.6: Own platform survival for hardkill and softkill with the LRTDP and reflex approaches.

Table 7.3: Average waiting time in seconds for the LRTDP and Tabu search approaches for hardkill weapons (standard deviation).

| $\|Ta\|$ | LRTDP | Tabu search |
|---|---|---|
| 1 | 1.027 (0.19) | 2.23 (0.40) |
| 2 | 1.81 (0.44) | 4.40 (0.74) |
| 3 | 2.79 (0.57) | 6.24 (1.12) |
| 4 | 3.02 (0.62) | 7.67 (1.48) |
| 5 | 3.64 (0.91) | 8.73 (1.75) |
| 6 | 4.50 (2.20) | 9.35 (1.93) |
| 7 | 5.11 (1.67) | 9.80 (1.93) |
| 8 | 7.35 (3.33) | 9.96 (2.00) |

Table 7.4: Average planning time for the LRTDP approach for hardkill weapons.

| $\|Ta\|$ | Average planning time in seconds (standard deviation) |
|---|---|
| 1 | 0.0016 (0.005) |
| 2 | 0.0079 (0.0083) |
| 3 | 0.04 (0.013) |
| 4 | 0.17 (0.269) |
| 5 | 1.39 (0.79) |
| 6 | 2.13 (0.72) |
| 7 | 4.87 (4.14) |
| 8 | 8.45 (7.38) |

of a scenario is in average 59.06 seconds, thus, these waiting times are not negligible. In particular, we can observe that the higher the number of threats a scenario has, the higher the waiting time is. This is because when there are many threats, the ship executes many allocations, which each has a waiting time.

Table 7.4 describes the average planning time in seconds for the LRTDP approach for hardkill weapons. Such a planning time is not prohibitive, even for a scenario with 8 threats. However, if the threats travel 2 times slower, the planning time augments drastically. For example, it takes 8 minutes to plan for a scenario with 6 threats in this case, instead of 2.13 seconds when the characteristics of the threats permit only 2 or 3 reengagements.

## 7.7 Discussion on Experiments using SADM

LRTDP has a certain number of advantages and disadvantages. In brief, the main advantage of LRTDP is that its policy is optimal. Its main disadvantage is that it

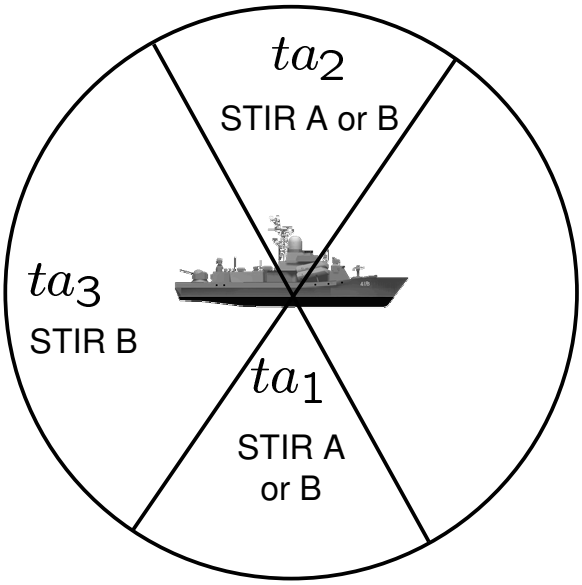



Figure 7.7: Example with 3 threats.

requires a precise model of the environment. This model also usually requires a large amount of states.

### 7.7.1 What it did well

LRTDP is able to converge to the optimal solution, if its model of the environment is accurate. Conversely, a greedy approach like the reflex one does not guarantee to converge to the optimal solution. To show that, let's consider the example in Figure 7.7 with three threats, which travel at the same speed. $ta_1$ is the nearest threat and can be engaged by a SAM with STIR A or B. $ta_2$ is the second nearest threat and can also be engaged by a SAM with STIR A or B. Finally, $ta_3$ is the furthest threat and it can only be engaged by a SAM with STIR B.

In this example, a reflex approach may allocate a SAM on $ta_1$ with STIR A and a SAM on $ta_2$ with STIR B. However, LRTDP would allocate a SAM on $ta_1$ with STIR B and a SAM on $ta_2$ with STIR A. Indeed, since $ta_1$ is nearer to the own platform, STIR B will become available earlier to counter threat $ta_3$, hence maximizing the possible reengagements on $ta_3$.

Let's consider another example where the minimal interceptions distance for the SAM is 2.2 km. There are three threats traveling at the same distance as our SAMs. The first threat $ta_1$ is at a range of 4.5 km and can be engaged with STIR A. The second threat $ta_1$ is at a range of 7 km and can be engaged with STIRs A and B. The third threat $ta_3$ is at a range of 9.5 km and can be engaged with STIRs A and B. Thus, under these characteristics, $ta_3$ can be engaged two times with a SAM. In this case, the

reflex planner engages a SAM on $ta_1$ with STIR A and a SAM on $ta_2$ with STIR B. Afterwards, we can engage a SAM on $ta_3$ with STIR A. Thus, a SAM can be engaged one time on each treat with the reflex planner.

The LRTDP planner would generate a better solution. Precisely, it can engage a SAM on $ta_1$ with STIR A and engage a SAM on $ta_3$ with STIR B. When $ta_1$ is intercepted, $ta_2$ has advanced of 2.2 km and is now at a range of 4.8 km. LRTDP can launch a SAM on $ta_2$ with STIR A. If the SAM on $ta_3$ missed it, is now at a range of 4.75 km. It can launch another SAM on $ta_3$ with STIR B. In this case, it can launch one SAM on $ta_1$ and $ta_2$. On the other hand, $ta_3$ can be engaged two times, which is a better policy than the one generated by a reflex planner.

In Figures 7.5 and 7.6, the LRTDP planner has a higher survivability for scenarios of 1 to 4 threats, which demonstrate the better solutions generated by LRTDP. As discussed in the next section, the LRTDP approach has a lower survivability than the reflex approach for scenarios of 5 to 8 threats is due to the very high waiting time which occurs in these scenarios.

### 7.7.2 What it did poorly

LRTDP has two major disadvantages. First of all, since it models the entire reachable state space, the planning time can be very prohibitive, as explained in Section 7.4.1. For this reason, the parameters of the scenarios should not cause a high number of possible reengagements for each threat. Also, as demonstrated in Table 7.3, the LRTDP approach is hindered by its high waiting time due to uncertain allocation durations. A reflex approach which does not have to wait when an allocation is finished is advantaged in this case. Indeed, LRTDP models the environment and reasons about it. If its model is not accurate, it will reason about fallacious information. Sometimes, it is better to have no model such as with the reflex approach than a fallacious model, as it is the case for LRTDP, with the uncertain action durations. We think the reason why the LRTDP approach has a lower platform survival than the reflex approach for scenarios of 5 to 8 threats in Figures 7.5 and 7.6 is due to the very high waiting times which occur in these scenarios. Indeed, a waiting time of 3.64 to 7.34 seconds is very important in scenarios that length 59.06 seconds in average. In our opinion, these waiting times cause the reflex planner to be more effective than LRTDP.

### 7.7.3 LRTDP Improvements

The most important improvement needed for LRTDP is to consider effectively uncertain action durations. A possible approach would be to discretize the action duration in many possible durations with an associated probability as done by Ramsauer (2002).

However, doing this would augment exponentially the branching factor from each state.

Another approach which might be adopted is the one that represents each task as an independent random variable with a known mean and variance Beck and Wilson (2007). They propose and investigate a number of techniques for solving such problem based on combination of Monte Carlo simulation, solutions to the associated deterministic problem, and either constraint programming or tabu search. Their empirical result demonstrate that a combination of the use of the associated deterministic problem and Monte Carlo simulation results in algorithms that scale best both in terms of problem size and uncertainty. These techniques should be investigated possibly in our future work to model the uncertain duration of our resource and tasks.

Another solution would be to generate a policy not only when a new threat is perceived, but also at each kill assessment. In this case, LRTDP would act at each events from the environment, like the reflex approach. However, these new planning points would also augment the planning time.

Dealing with uncertain action durations seems an active research area where it could be interesting determining a viable solution for our problem.

### 7.7.4 Discussion on Target Problem

In Section 4.3.2, we have described the specific problem we are interested in to solve using SADM. The implemented softkill planner described in Section 7.3.1 tried to model the engagements with respect with the target problem and with the kill probabilities learned in SADM. However, the movement has not been considered in this work because of the continuous action space it involves. Still, we could discretize the action space for the movement like we did in previous works (Blodgett et al., 2002).

## 7.8 Conclusion

This chapter described the implementation of a reflex and LRTDP approaches in the SADM simulator. The reflex approach, which is based on simple rules, is more efficient than the LRTDP approach on scenario with 5 to 8 threats. However, the LRTDP approach is better in scenarios with 1 to 4 threats. We conjectured that the reason why the LRTDP planner is worse than the reflex planner in scenario with many threats is due to the high waiting time induced by the uncertain duration of the actions. Considering effectively these high waiting time for LRTDP is for future work. We now conclude this thesis with a summary of the main contributions. Also, possible future works are proposed.

# Chapter 8

# Conclusion

This thesis considers the problem of resource allocation for domains where the preferences of agents for the resources are induced by stochastic-planning problems (modeled as Markov decision processes). We demonstrated that, by considering the specific structure of our problem, and by defining tight initial bounds for resource allocation the planning complexity is greatly reduced.

In this chapter, we conclude with a summary of the main contributions of the work, along with a discussion of its limitations and an overview of future directions and open problems.

## 8.1 Summary of the Contributions

Conceptually, this dissertation can be broken into two parts. The first part, consisting of Chapters 5 and 7, discusses policy-optimization and resource allocation problems using real-time heuristic search. The second part, which corresponds to Chapter 6 deals with models involving exploiting the task creation, and resource interactions of the problem, and problem structure. The overarching goal of these two parts is on developing efficient planning and resource-allocation algorithms by exploiting problem structure due to the specific characteristics of our problem. Below, we briefly summarize the main results and contributions of this work, with an emphasis on the connections between the parts.

- **Exploiting the task creation, and resource interactions of the problem** This contributions consists of two decomposition techniques which exploits the characteristic of the problem. Firstly, the acyclic decomposition in Section 6.2 exploits the task creation by current tasks in the problem to generate an acyclic graph of cyclic components. This graph is solved from the leafs to the root and permits a considerable reduction in planning time. Furthermore, an agent

may manage each resource type as in the MTAMDP approach. In this case, positive and negative interactions among resources of distinct agents as well as simultaneous actions are coordinated near-optimally through a *central* agent. We also presented the extensibility of the acyclic decomposition and merged it to the MTAMDP approach to further reduce the planning time.

- **policy-optimization and resource allocation problems using real-time heuristic search.** A possible type of resource allocation problem is where the resources are already shared among the agents, but the actions made by an agent may influence the reward obtained by at least another agent. In this case, Q-decomposition can be employed to generate an optimal policy since each agent can compute a Q-value from its perspective. Then an arbitrator sums the Q-value of each agents for each possible joint actions. If an action is not possible to execute due to resource constraint, this global action is discarded. This Q-decomposition model is used in the context of real-time heuristic search and permits to reduce significantly the complexity to formulate an optimal policy for the agents.

  However, when the resources are available to all agents, no Q-decomposition is possible. In this case, tight bounds for real-time dynamic programming permits to focuss the search on relevant states, and to prune the actions in a state which their upper bound Q-values are lower than the lower bound of the state.

## 8.2 Future Works

This dissertation may be extended in many ways to solve more effectively our resource allocation problem. Below, we briefly outline a few of the possible directions of future work and discuss preliminary ideas on pursuing these topics.

### 8.2.1 Partially Observable Environment

As discussed in Section 4.3.2, the environment of a ship, which has to counter anti-ship missile using its resources is partially observable. Indeed, the sensors of the ship perceive an imperfect model of the environment. In this case, a Partially Observable Markov Decision Process (POMDP) (Smallwood and Sondik, 1973) is a convenient tool to model such a problem. A drawback of POMDPs is its huge complexity, but researches in this field are very active.

An interesting algorithm which could be extended with the work of this thesis is Heuristic Search Value Iteration (HSVI) (Smith and Simmons, 2004). HSVI draws on prior approaches that combine heuristic search and value iteration (Washington (1997);

Geffner and Bonet (1999)), and a multitude of algorithms that employ a piecewise linear convex value function representation and gradient backups (Cassandra et al. (1997); Pineau et al. (2003)). HSVI keeps track of upper and lower bounds on the value function. When HSVI is used to solve a resource allocation problem like ours, the initial bounds could be initialized with a POMDP version of the upper and lower bounds defined in Section 5.2. In addition, the acyclic decomposition and the Q-decomposition could be applied in a partially observable environment also.

Furthermore, Real-Time Belief State Search (RTBSS) (Paquet et al., 2005) is an on-line POMDP algorithm which finds effective solutions in a very short amount of time. It has been shown that RTBSS could be improved when initialized with the solution of an off-line approach, such as HSVI and PBVI by Pineau et al. (2003). Again, our tight bounds, the acyclic decomposition, and Q-decomposition could speed up the convergence of RTBSS, which could enable a deeper search and a better policy.

### 8.2.2 Decentralized Problem

This thesis considered the resource allocation of a single ship which has to execute many tasks (counter many threats) with its limited resources. In these days, current war scenarios usually involves multiple combat platforms which should coordinate themselves to generate a joint policy. Such a decentralize problem has the effect that every actor perceive a separate part of the environment and the global view of the environment is not so obvious.

Such problems, can be modeled as a Multi-agent Markov Decision Process(MMDP) proposed by Boutilier (1999), the Partially Observable Identical Payoff Stochastic Game (POIPSG) proposed by Peshkin et al. (2000), the multi-agent decision process proposed by Xuan and Lesser (2002), the Communicative Multiagent Team Decision Problem (COM-MTDP) proposed by Pynadath and Tambe (2002), the Decentralized Markov Decision Process (DEC-POMDP and DEC-MDP) proposed by Bernstein et al. (2002), and the DEC-POMDP with Communication (DEC-POMDP-COM) proposed by Goldman and Zilberstein (2003).

The MMDP model is based on full observability of the global state by each agent. However, multiple combat platforms introduce situations in which each agent has a different partial view of the global state and together they can uniquely determine it. This observation model is characteristic of a DEC-MDP, DEC-POMDP and the Xuan-Lesser framework. The other models differ in that they assume a more general form of observations where there is joint partial observability.

Another interesting approach is the Multi-Agent A* (MAA*) approach for solving a DEC-POMDP from Szer et al. (2005). All these approaches should be examined to determine if the algorithms proposed in this thesis could extend these works and

diminish the complexity to formulate a policy in such a complex environment.

### 8.2.3 Structured Methods for Consumable Resources

The state space in this thesis represents all combination of possible available consumable resources, which is very complex. However, in Section 6.1, if the resources are already shared among the agents, but the actions made by an agent may influence the reward obtained by at least another agent, we can use Q-decomposition which diminishes greatly the complexity associated to the consumable resources.

A similar framework to Q-decomposition are Progressive Reasoning Units (Mouaddib and Zilberstein, 1998) (PRUs). A (PRU consists of decomposing a global task in many sub tasks to be executed in sequences. This permits to decompose the set of resource to consider. For example, the problem describes in Section 4.3.2 can be modeled efficiently using a PRU. Indeed, when the threat is searching for the ship or when it is locked requires different resource types, that can be model using PRUs.

### 8.2.4 Continuous State and Action Spaces

As pointed out in Section 7.7.2 uncertain action durations introduce a continuous action time. In this case, when the action duration is approximated, the resulting policy introduces waiting time between actions. We described possible two possible ways to deal with this problem in Section 7.7.3, which are to discretize the action duration in many possible durations and to introduce more planning points.

Other actions are continuous, such as the movement, which is depicted in Figure 7.4. The number of possible movement actions are infinite. Again one way to tackle this is to discretize the action space or to deal with the continuous action space. However dealing with the continuous action space would augment drastically the computing time. Dealing with continuous state and action spaces for our real-time problem seems a very attractive research area.

# Bibliography


Aberdeen, D. and Baxter, J. (2002). Scaling internal-state policy-gradient methods for POMDPs. In *Proceedings of the Nineteenth International Conference on Machine Learning*, Sydney, Australia.

Aberdeen, D., Thiebaux, S., and Zhang, L. (2004). Decision-theoretic military operations planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-04)*, Whistler, Canada.

Ahuja, R. K., Kumar, A., and Jha, K. (2003). Exact and heuristic methods for the weapon target assignment problem. *Social Science Research Network Electronic Library*.

Altman, E. (1999). *Constrained Markov Decision Processes*. Chapman and HALL/CR, Cambridge, Massachusetts.

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138.

Beck, J. C. and Wilson, N. (2007). Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*, 28:183–232.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.

Bernstein, D., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27(4):819–840.

Berry, P. M. (1993). Uncertainty in schedulingl probability, problem reduction, abstractions, and the user. *IEE Colloquium on Advanced Software Technologies for Scheduling*, (193/163).

Bertsekas, D. (2005). Rollout algorithms for constrained dynamic programming. Technical report 2646, Lab. for Information and Decision Systems, MIT, Mass., USA.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Besse, C., Cinq-Mars, P., Gagné, O., Chouinard, S., Chaib-draa, B., Benaskeur, A., and Blodgett, D. (2007). A multi-agent coordination algorithm for weapon assignment in ship self defense. Unpublished technical report, Computer Science and Software Engineering Department, Laval University, Quebec, Canada.

Besse, C., Plamondon, P., Chaib-draa, B., Benaskeur, A., and Blodgett, D. (2006). R-FRTDP: A real-time DP algorithm with tight bounds for a stochastic resource allocation problem. In *Proceedings of the 20th Canadian Conference on Artificial Intelligence (CAI-2007)*, Montréal, Canada.

Beynier, A. and Mouaddib, A. I. (2006). An iterative algorithm for solving constrained decentralized markov decision processes. In *Proceeding of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*.

Blodgett, D., Plamondon, P., Chaib-draa, B., Kropf, P., and Boss, E. (2002). A method to optimize ship maneuvers for the coordination of hardkill and softkill weapons within a frigate. In *Proceedings of The $7^{th}$ International Command and Control Research and Technology Symposium (ICCRTS-2002)*, Quebec, QC.

Blodgett, D. E., Gendreau, M., Guertin, F., Potvin, J.-Y., and Séguin, R. (2003). A tabu search heuristic for resource management in naval warfare. *Journal of Heuristics*, 9(2):145–169.

Bonet, B. and Geffner, H. (2001). Planning and control in artificial intelligence: A unifying perspective. *Applied Intelligence*, 14(3):237–252.

Bonet, B. and Geffner, H. (2003a). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-2003)*.

Bonet, B. and Geffner, H. (2003b). Labeled RTDP: Improving the convergence of real-time dynamic programming. *Proceeding of the 13Th International Conference on Automated Planning & Scheduling (ICAPS-2003)*, pages 12–21.

Boukhtouta, A. (2002). Planning military operations under uncertainty. Technical report TR 2001-221, Decision support systems section, Canada.

Boutilier, C. (1999). Sequential optmiality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 478–485, Stockholm.

Boutilier, C., Brafman, R. I., and Geib, C. W. (1997). Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1156–1162, Stockholm.

Boutilier, C., Dean, T., and Hanks, S. (1999a). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94.

Boutilier, C., Dearden, R., and Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107.

Boutilier, C. and Goldszmidt, M. (1996). The frame problem and bayesian network action representations. In *Proceedings of the Eleventh Biennal Canadian Conference on Artificial Intelligence (CAI-1996)*, pages 69–83, Toronto.

Boutilier, C., Goldszmidt, M., and Sabatag, B. (1999b). Sequential auctions for the allocation of resources with complementarities. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 527–534, Stockholm.

Bowling, M. and Veloso, M. (2004). Existence of multiagent equilibria with limited agents. *Journal of Artificial Intelligence Research*, 22:353–384.

Boyd, J. R. (1987). Organic design for command and control. Briefing slides.

Braford, J. C. (1961). Determination of optimal assignment of a weapon system to several targets. AEREITM-9, Vought Aeronotics, Dallas, Texas.

Braziunas, D. and Boutillier, C. (2004). Stochastic local search for POMDP controllers. In *Proceedings of AAAI 2004*, San Jose, CA.

Brown, C., Fagan, P., Hepplewhite, A., Irving, B., Lane, D., and Squire, E. (2001). Real-time decision support for the anti-air warfare commander. In *$6^{th}$ International Command and Control Research and Technology Symposium (ICCRTS-2001)*, Annapolis, Maryland.

Burns, A., Punnekkat, S., Littlewood, B., and Wright, D. W. (1997). Probabilistic guarantees for fault-tolerant real-time systems. Technical report deva tr no. 44, design for validation, esprit long term research project no. 20072. available at http://www.fcul.research.ec.org/deva.

Cassandra, A. R., Littman, M. L., and Zhang, N. (1997). Incremental pruning: A simple, fast, exact method for partially observable markov decision processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-1997)*.

Castanon, D. A. (1987). Advanced weapon-target assignment algorithm quaterly report. Tr-337, ALPHA TECH Inc., Burlington, Massachusetts.

Chaib-draa, B., Paquet, S., and Plamondon, P. (2001). Proposal #222802-98: Resource management in complex sociotechnical systems: Report on first stage. Technical report, Laval University, Qc.

Chalmers, B. A. (1995). Toward time-dependent planning of missile allocation and engagement scheduling in a naval threat evaluation and weapon assignment (TEWA) system. Technical report AN-95-00511, Defence Research Establishment Valcartier, Situation assessment and resource allocation group, Valcartier, Québec, Canada, Qc.

Chalmers, B. A. (1997). Design issues for a decision support systems for a modern frigate. In *Proceedings of the 2nd Annual Symposium and Exhibition on Situational Awareness in the Tactical Air Environment*, Naval Air Warfare Center, Patuxent River, Maryland.

Chang, S. C., James, R. M., and Shaw, J. J. (1987). Assignment algorithm for kinetic energy weapons in boost defence. In *Proceedings of the IEEE 26th Conference on Decision and Control*, Los Angeles, California.

Daniels, R. L. and Carrillo, J. E. (1997). $\beta$-robust scheduling for single-machine systems with uncertaint precessing times. *IIE Transactions*, 29:977–985.

Davenport, A. and Beck, J. (2002). A survey of techniques for scheduling with uncertainty. Unpublished manuscript.

Day, R. H. (1966). Allocating weapons to target complexes by means of nonlinear programming. *Operation Research*, 14:992–1013.

Dean, T. and Lin, S. (1995). Decomposition techniques for planning in stochastic domains. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence (IJCAI-1995)*, San Francisco.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of a*. *Journal of the ACM*, 32(3):505–536.

Decker, K. S. and Lesser, V. R. (1992). Generalizing the partial global planning algorithm. *International Journal of Intelligent Cooperative Information Systems*, 1(2):319–346.

DenBroader, G. G., Ellison, R. E., and Emerling, L. (1958). On optimum target assignments. *Operation Research*, 7:322–326.

Dolgov, D. A. and Durfee, E. H. (2004). Optimal resource allocation and policy formulation in loosely-coupled markov decision processes. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-2004)*, pages 315–324.

Draper, D., Hanks, S., and Weld, D. (1994). Probabilistic planning with information gathering and contingent execution. In Hammond, K., editor, *Proceedings of the*

*Second International Conference on AI Planning Systems (ICAPS-1994)*, pages 31–36, Menlo Park, California. American Association for Artificial Intelligence.

Drummond, M., Bresina, J., and Swanson, K. (1994). Just-in-case scheduling. In Press, A. P., editor, *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1098–1104, Menlo Park, CA.

Eckler, A. R. and Burr, S. A. (1972). Mathematical models of target coverage and missile allocation. *Military Operation Research Society*.

Feinberg, E. A. and Shwartz, A. (1996). Constrained discounted dynamic programming. *Mathematics of Operations Research*, 21:922–945.

Feldmann, R., Gairing, M., Lucking, T., Monien, B., and Rode, M. (2003). Selfish routing in non-cooperative networks: A survey. In *28th International Symposium on Mathematical Foundations of Computer Science (MFCS-2003)*, pages 21–45. Springer-Verlag.

Frini, A., Guitouni, A., Benaskeur, A., and Bossé, E. (2004). Multiple criteria dynamic allocation of shipboard weapons. In *Information Systems and Management Sciences (MCO 2004)*, Metz, France.

Garey, M. R. and Johnson, D. S. (1979). *Computer and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Garvey, A., Humphrey, M., and Lesser, V. (1993). Task interdependencies in design-to-time real-time scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-1993)*, pages 580–585.

Garvey, A. and Lesser, V. (1993). A survey of research in deliberative real-time artificial intelligence. Technical report 93-84, Massachusetts University, Computer Science Department.

Geffner, H. and Bonet, B. (1999). Solving large pomdps by real time dynamic programming. In *Working Notes Fall AAAI Symposium on POMDPs*.

Ghose, D. (2003). On the generalisation of true proportional navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 30(2):545–555.

Glazebrook, K. and Washburn, A. R. (2004). Shoot-look shoot: A review and extension. *Operations Research*, 52:454–463.

Goldman, C. V. and Zilberstein, S. (2003). Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003)*, Melbourne, Australia.

Grant, K. E. (1993). Optimal resource allocation using genetic algorithm. *Naval Review*, pages 174–175.

Green, D. J., Moore, J. T., and Borsi, J. J. (1997). An integer solution heuristic for the arsenal exchange model (AEM). *Military Operations Research Society*, 3(2).

Guestrin, C., Koller, D., and Parr, R. (2001). Solving factored POMDPs with linear value functions. In *IJCAI-01 Workshop on Planning under Uncertainty and Incomplete Information*, Seattle.

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. (2003). Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468.

Hansen, E. A. (1998). Solving pomdps by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 211–219, Madison, Wisconsin.

Hansen, E. A. and Feng, Z. (2001). Approximate planning for factored pomdps. In *Proceedings of the Sixth European Conference on Planning (ECP-01)*, Toledo, Spain.

Hart, E. and Ross, P. (1999). An immune system approach to scheduling in changing environment. In W. Banzhaf, Daida, A. E. E. M. H. G. V. H. M. J. and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionnary Computation Conference (GECCO-99)*, pages 1559–1565. Morgan Kaufman.

Herroelen, W. S. and Demeulemeester, E. L. (1995). Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems. In P. Chretienne, E. G. Coffman Jr., J. K. L. and Liu, Z., editors, *Scheduling theory and its applications*. Chapter 12, John Wiley & Sons, Ltd.

Hildum, D. W. (1994). *Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems*. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway.

Hong, X., Wilson, P. A., and Harris, C. J. (2003). A line of sight counteraction navigation algorithm for ship encounter collision avoidance. *Journal of Navigation*, 56:111–121.

Hosein, P. A. and Athans, M. (1990). Some analytical results for the dynamic weapon-target allocation problem. *Naval Research Logistics*.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts.

Jennings, N. R. (1994). *Cooperation in industrial multi-agent systems*. World Scientific Publishing, River Edge, NJ, USA.

Kaelbling, L. P., Littman, M., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134.

Katter, J. D. (1986). A solution of the multi-weapon, multi-target assignment problem. Working paper 26957.

Kearns, M., Littman, M. L., and Singh, S. (2001). Graphical models for game theory. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI01)*, pages 253–260.

Koller, D. and Milch, B. (2003). Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1):181–221.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence*, 42(3):189–211.

Laroche, P., Boniface, Y., and Schott, R. (2001). A new decomposition technique for solving markov decision processes. In *Proceedings of the 2001 ACM Symposium on Applied Computing*, pages 12–16. ACM Press.

Leon, V. J., Wu, S. D., and Storer, R. H. (1994). Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43.

Liang, T. (1995). Getting the act together. In *International Defence Review*, Netherland.

Liang, T. and Liem, K. D. (1992). Integrated naval air defence coordinating hardkill and softkill weapons. In *International Defence Review*, Netherland.

Lloyd, S. P. and Witsenhausen, H. S. (1986). Weapons allocation is NP-complete. In *Proceedings of the 1986 Summer Computer Simulation Conference*, Reno, Nevada.

Mailler, R., Lesser, V., and Horling, B. (2003). Cooperative Negotiation for Soft Real-Time Distributed Resource Allocation. In *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, pages 576–583, Melbourne. ACM Press.

Majercik, S. M. and Littman, M. L. (2003). Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147(1-2):119–162.

Maltin, S. M. (1970). A review of the literature on the missile-allocation problem. *Operation Research*, 18:334–373.

Manne, A. S. (1958). A target-assignment problem. *Operations Research*, 6:346–351.

Martelli, A. and Montanari, U. (1978). Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12):1025–1039.

Mas-Colell, A., Whinston, M. D., and Green, J. R. (1995). *Microeconomic Theory*. Oxford University Press, New York.

Mausam, Benazera, E., Brafman, R., Meuleau, N., and Hansen, E. A. (2005). Planning with continuous resources in stochastic domains. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1244 – 1252, Edinburgh, Scotland.

McAfee, R. P. and McMillan, J. (1996). Analyzing the airwaves auction. *Journal of Economic Perspectives*, 10(1):159–175.

McMahan, H. B., Likhachev, M., and Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the Twenty-Second International Conference on Machine learning (ICML-2005)*, pages 569–576, New York, NY, USA. ACM Press.

McMillan, J. (1994). Selling spectrum rights. *Journal of Economic Perspectives*, 8(3):145–162.

Metler, W. A. and Preston, F. L. (1990). A suite of weapon assignment algorithms for a sdi mid-course battle manager. NRL memorandum report 671, Naval Research Laboratory, Washington, DC.

Meuleau, N., Hauskrecht, M., Kim, K., Peshkin, L., Kaebling, L. P., Dean, T., and Boutilier, C. (1998). Solving very large weakly coupled markov decision processes. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-1998)*, pages 165–172. AAAI Press.

Meuleau, N., Peshkin, L., Kim, K., and Kaelbling, L. P. (1999). Learning finite-state controllers for partially observable environments. In *Proceedings of the Fourtheen Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 427–436, Stockholm.

Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

Morissette, J.-F., Chaib-draa, B., and Plamondon, P. (2004). Resource allocation in time-constrained environments: The case of frigate positioning in anti-air warfare. In *Information Systems and Management Sciences (MCO 2004)*, Metz, France.

Mouaddib, A. and Zilberstein, S. (1998). Optimal scheduling of dynamic progressive processing. In *European conference on artificial intelligence (ECAI)*, pages 499–503.

Murphey, R. A. (1999). Target-based weapon target assignment problems. In Pardalos, P. M. and Pitsoulis, L. S., editors, *Nonlinear Assignment Problems: Algorithms and Applications*, pages 39–53, Alexandra, Virginia. Kluwer Academic.

Musliner, D., Durfee, E., Goldman, R., Boddy, M., Wu, J., and Dolgov, D. (2006). Coordinated plan management using multiagent mdps. In *AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*, Palo Alto, California.

Nilsson, N. J. (1980). *Principles or Artificial Intelligence*. Tioga Publishing, Palo Alto, Ca.

Nuitjen, W. P. M. and Aarts, E. H. L. (1997). A computation al study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*.

Oard, D. W., Wolk, S. I., and Ephremides, A. (1994). On the integrated scheduling of hardkill and softkill assets using dynamic programming. Technical report NRL-FR-5750-94-9721, Naval Research Laboratory, Military Support Division, Washington, United States.

Orlin, D. (1987). Optimal weapons allocation against layered defenses. *In Naval Research Logistics*, 34.

P. Hart, N. Nilsson, B. R. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Science and Cybernetics*, 4(2):100–107.

Pape, C. L. (1991). Constraint propagation in planning and scheduling. Cife technical report, Robotic Laboratory, Deptartment of COmputer Science, Stanford University.

Paquet, S., Tobin, L., and Chaib-draa, B. (2005). An online pomdp algorithm for complex multiagent environments. In *of The fourth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-05)*, Utrecht, The Netherlands.

Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. (2000). Learning to cooperate via policy search. In *Proceedings of the Sixteenth International Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pages 489–496.

Pindyck, R. S. and Rubinfeld, D. L. (2000). *Microeconomics*. Prentice Hall.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2003)*.

Plamondon, P. (2003). A frigate survival approach based on real-time multiagent planning. Master's thesis, Computer Science & Software Engineering Department, Laval University.

Plamondon, P., Chaib-draa, B., Beaumont, P., and Blodgett, D. (2003). A frigate movement survival agent-based approach. In *Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES-2003)*. University of Oxford, United Kingdom.

Plamondon, P., Chaib-draa, B., and Benaskeur, A. (2005). Decomposition techniques for a loosely-coupled resource allocation problem. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2005)*, Compiègne, France.

Plamondon, P., Chaib-draa, B., and Benaskeur, A. (2006a). An efficient resource allocation approach in real-time stochastic environment. In *Proceedings of the 19th Canadian Conference on Artificial Intelligence (CAI-2006)*, Québec, Canada.

Plamondon, P., Chaib-draa, B., and Benaskeur, A. (2006b). A multiagent task associated mdp (mtamdp) approach to resource allocation. In *AAAI 2006 Spring Symposium on Distributed Plan and Schedule Management*, Palo Alto, California.

Plamondon, P., Chaib-draa, B., and Benaskeur, A. (2006c). A q-decomposition lrtdp approach to resource allocation. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-2006)*, Hong Kong, China.

Plamondon, P., Chaib-draa, B., and Benaskeur, A. (2007a). A q-decomposition and bounded rtdp approach to resource allocation. In *Proceedings of the Sixth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2007)*.

Plamondon, P., Chaib-draa, B., and Benaskeur, A. (2007b). A real-time dynamic programming decomposition approach to resource allocation. In *Proceedings of the Information, Decision and Control 2007 Conference (IDC-2007)*, Adelaide, Australia.

Plamondon, P., Chaib-draa, B., and Benaskeur, A. (2007c). Tight bounds for a stochastic resource allocation algorithm using marginal revenue. In *AAAI 2007 Spring Symposium on Game Theoretic and Decision Theoretic Agents*, Palo Alto, California.

Poupart, P. and Boutilier, C. (2003). Bounded finite state controllers. In *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada.

Poupart, P. and Boutilier, C. (2004). VDCBPI: an approximate scalable algorithm for large scale POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, Vancouver.

Precup, D. and Sutton, R. S. (1998). Multi-time models for temporally abstract planning. *Advances in Neural Information Processing Systems 10*.

Pryor, L. and Collins, G. (1993). Cassandra: Planning for contingencies. Technical Report 41, Northwestern University, The Institute for the Learning Sciences.

Pynadath, D. and Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, pages 389–423.

Ramsauer, M. (2002). Design-to-time scheduling with hard real-time constraints and non-deterministic execution time. Unpublished technical report, Faculty of Mathematics and Informatics, University of Passau, Germany.

Roy, N. and Gordon, G. (2002). Exponential family PCA for belief compression in POMDPs. In *Advances in Neural Information Processing Systems*, pages 1635–1642, Vancouver, Canada.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical report CUED/FINFENG/TR 166, Cambridge University Engineering Department.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs.

Russell, S. J. and Subramanian, D. (1995). Provably bounded optimal agents. *Journal of Artificial Intelligence Research*, 2:575–609.

Russell, S. J. and Zimdars, A. (2003). Q-decomposition for reinforcement learning agents. In *ICML*, pages 656–663.

Scherrer, B. (2004). Neurocomputing for optimal control and reinforcement learning with large state space. *Neurocomputing (to appear)*.

Sheffi, Y. (2004). Combinatorial auctions in the procurement of transportation services. *Interfaces*, 34(4):245–252.

Singh, S. P. and Cohn, D. (1998). How to dynamically merge markov decision processes. *Advances in Neural Information Processing Systems 10*.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088.

Smith, D. E., Frank, J., and Jonsson, A. K. (2000). Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(11):268–272.

Smith, T. and Simmons, R. (2004). Heuristic search value iteration for POMDPs. In *Proceedings of Uncertainty in Artificial Intelligence (UAI-2004)*.

Smith, T. and Simmons, R. (2006). Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*, Boston, USA.

Song, J. and Regan, A. (2002). Combinatorial auctions for transportation service procurement: The carrier perspective. *Transportation Research Record*, 1833:40–46.

Soucy, M. (2003). Planification déliberative en temp réel. Master's thesis, Computer Science & Software Engineering Department, Laval University.

Spaan, M. . T. and Vlassis, N. A. (2004). Point-based POMDP algorithm for robot planning. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA.

Szer, D., Charpillet, F., and Zilberstein, S. (2005). MAA*: A heuristic search algorithm for solving decentralized pomdps. In *Proceedings of the 21th Conference on Uncertainty in Artificial Intelligence (UAI-2005)*.

Tarjan, R. E. (1972). Depth first search and linear graph algorithm. *SIAM Journal on Computing*, 1(2):146–172.

Visser, D. (2001). EWTDA' the electronic warfare tactical decision aid for the aor and lpd. A TNO Physics and Electronics Laboratory Publication.

Wacholder, E. (1989). A neural network-based optimization algorithm for the static weapon-target assignment problem. *ORSA Journal on Computing*, 4:232–246.

Waltz, E. L. and Buede, D. M. (1986). Data fusion and decision support for command and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(6):865–879.

Washington, R. (1997). BI-POMDP: Bounded, incremental, partially-observable Markov-model plannings. In *In proceedings of the European Conference on Planning (ECP-1997)*, Toulouse, France.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

Wellman, M. P., Walsh, W. E., Wurman, P. R., and MacKie-Mason, J. K. (2001). Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303.

Wellman, W. P. (1993). Challenges of decision-theoretic planning. In *Proceedings of the AAAI Spring Symposium on Fundations of Automatic Planning*.

Wilkins, D. E., Myers, K. L., Lowrance, J. D., and Wesley, L. P. (1995). Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227.

Wu, C. C. and Castanon, D. A. (2004). Decomposition techniques for temporal resource allocation. Technical report: AFRL-VA-WP-TP-2004-311, Air Force Research Laboratory, Air force base, OH.

Wurman, P. R. and Wellman, M. P. (1996). Optimal factory scheduling using stochastic dominance a*. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence (UAI-96)*.

Xuan, P. and Lesser, V. (2002). Multi-agent polices: From centralized ones to decentralized ones. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*, Bologna, Italy.

Zhang, W. (2002). Modeling and solving a resource allocation problem with soft constraint techniques. Technical report: WUSC-2002-13, Washington University, Saint-Louis, Missouri.

Zweben, M. and Fox, M. S. (1994). *Intelligent Scheduling*. Morgan Kaufmann.

# Appendix A

# AND/OR Graphs

The most general state-space search problem considered in the AI literature is AND/OR graph search (Martelli and Montanari (1978); Nilsson (1980)). An AND/OR graph can be defined as a hypergraph which has *k-connectors* that connect a state to a set of $k$ successor states. Figure A.1 relates the concept of OR and AND nodes to the concept of a hyperarc or $k$-connectors. Figure A.1 (a) shows an OR node with two arcs leaving it, one for action $a_1$ and one for action $a_2$. Each arc leads to an AND node with two successor OR nodes, one for each possible successor state. (By convention, a square denotes an OR node and a circle denotes an AND node. In the terminology of decision analysis, a square corresponds to a choice node and a circle corresponds to a chance node.). Figure A.1 (b) shows a state, indicated by a circle, with two 2-connectors leaving it, one for action $a_1$ and one for action $a_2$. Each 2-connectors leads to two possible successor states. The representation on the right, using state nodes and $k$-connectors, is equivalent to the representation on the left, which uses OR and AND nodes.

A $k$-connectors can be interpreted in different ways. In problem-reduction search, it is interpreted as the transformation of a problem into $k$ subproblems. We use $k$-connector to consider the problem of planning under uncertainty in which it is interpreted as an action with an uncertain outcome. The action transforms a state into one of $k$ possible successor states, with a probability attached to each successor. In AND/OR graph search, a solution takes the form of an acyclic subgraph called a solution graph, which is defined as follows:

- The start state belongs to a solution graph.
- For every nongoal state in a solution graph, exactly one outgoing $k$-connector (corresponding to an action) is part of the solution graph and each of its successor states also belongs to the solution graph.
- Every directed path in the solution graph terminates at a goal state.

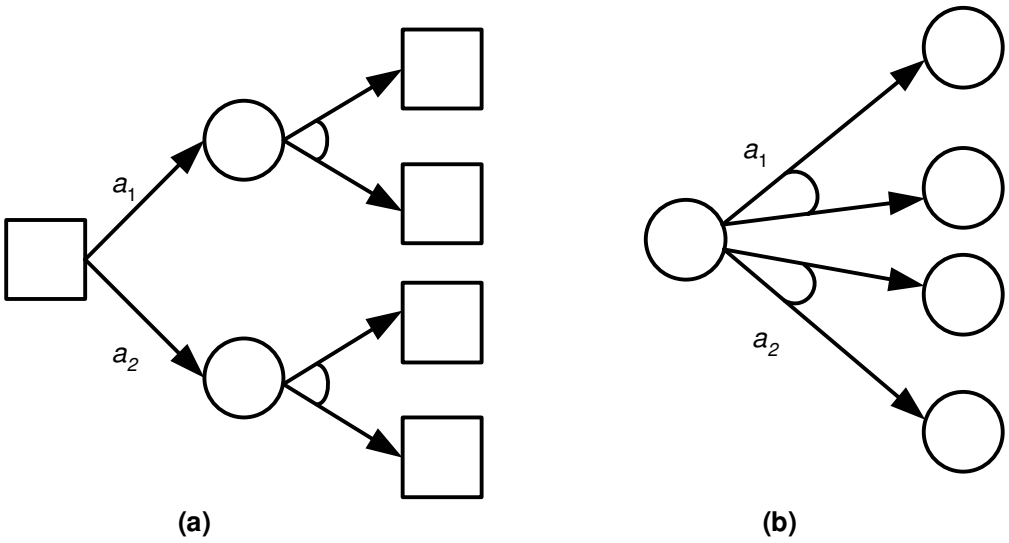



Figure A.1: (a) uses OR and AND nodes; and (b) uses state nodes and $k$-connectors.

AND/OR graphs are the basic tool used by the AO$^*$ and LAO$^*$ algorithms.