12TH ICCRTS "Adapting C2 to the 21st Century"

Modeling and Simulation
C2 Concepts, Theory, and Policy
Network-Centric Experimentation and Applications

Toward Efficient Optimal Resource Allocation for a Frigate

Pierrick Plamondon*, Brahim Chaib-draa
Laval University Québec, PQ, Canada, G1K 7P4
(418) 656-2131 x3226
Computer Science Department
{plamon; chaib}@damas.ift.ulaval.ca

Abder Rezak Benaskeur
Defence R&D Canada — Valcartier
2459 Pie-XI Blvd. North, Val-Bélair PQ, Canada, G3J 1X5
(418) 844-4000 x4478
Decision Support Systems Section
abderrezak.benaskeur@drdc-rddc.gc.ca

* Point of contact and student author

Abstract

The allocation of anti-air warfare weapon is an important aspect of command and control for a Frigate. This paper contributes to solve effectively the stochastic resource allocation problems known to be NP-Complete. Indeed, recently, efficient resource allocation algorithms have been developed in Artificial Intelligence which could improve adapt C2 to the 21st century. To address this complex resource management problem the Labeled Real-Time Dynamic Programming (LRTDP) approaches is applied in an effective

way. LRTDP concentrates the planning on significant states of the environment only, as the search is guided by an initial heuristic. As demonstrated by the experiments, LRTDP permits to obtain a very high survivability for the frigate, when compared to two other well known techniques.

1 Introduction

The Combat System of a typical Frigate includes above water warfare (AWW) weapon systems for hardkill and softkill. Increasing complexity in threat technology, and increasing speed and diversity in open-ocean and littoral threat scenarios makes efficient and effective planning for weapons resources more and more difficult. To counter these problems, research is ongoing to design and implement resource management decision aids, based on intelligent agent technology to perform AWW hardkill and softkill resource allocations scheduling for a Frigate.

This paper 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. 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.

A common way of addressing this large stochastic problem 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* Hansen and Zilberstein [2001] are all state-of-the-art heuristic search approaches in a stochastic environment. Because of its anytime quality, an interesting approach is RTDP introduced by Barto *et al.* Barto *et al.* [1995] which updates states in trajectories from an initial state s_0 to a goal state s_g. RTDP is used in this paper to solve efficiently a constrained resource allocation problem.

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 bounded RTDP 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. The efficient state trajectories updates made by state-of-the-arts BRTDP and FRTDP are used here, are reduced in numbers when given tight admissible bounds for our resource allocation problem. Also, we propose to prune the action space of BRTDP and FRTDP as in the approach of Singh and Cohn [1998].

The bounds proposed by Singh and Cohn [1998] are suitable to resource allocation, and are extended in this paper, using in particular the concept of *marginal revenue* Pindyck and Rubinfeld [2000]. Our bounds are compared theoretically and empirically to the ones proposed by Singh and Cohn. The problem is now modelled.

2 Problem Formulation

Our problem of interest is military naval operations which are known to be very complex. In this context, a ship's Commanding Officer needs to set his resources to maximum efficiency in real-time where he is in face to multi-threats situations. An efficient resource allocation can increase the chance of survival of the ship. In the case of C2 Above-Water Warfare (AWW), the list of main operations are as follows :

- *Threat detection*: Based on data from several sensors.
- *Resource allocation*: Resources are assigned to engage each threat.
- *Engagement control*: The process by which decisions in the two preceding steps are executed in real-time.

In our concern, we address how we develop a decision support system system that focusses specifically on some particular aspects of the C2, in order to reduce the complexity of the domain. Here, the focuss is on the *Resource allocation* and *engagement control* processes. To this end, the *threat detection* is considered as a black box. Not working on threat detection reduces the large volume of data that needs to be processed, which helps reducing the system's complexity to focuss on the resource allocation and execution parts.

The AWW hardkill weapons are weapons that are directed to intercept a threat and actively destroy it through direct impact or explosive detonation in the proximity of the threat. The range of different types of hardkill weapons varies, and the effectiveness of these weapons depends on a variety of factors, like distance to the threat, type of threat, speed of the threat, environment, etc. The AWW hardkill weapons for a typical Frigate include surface-to air missiles (SAMs) that have the greatest range, an intermediate range Gun, and a Close-In Weapons System (CIWS) that is a short-range, rapid-fire gun. The Gun has a blind zone of ± 35 deg at the back of the Frigate. Closely allied to these weapons are two Separate Tracking and Illuminating Radars (STIRs) that are used to guide a SAM to a threat, and to point the Gun. There is one STIR to the front and one STIR to the back of the ship. In his case, both stirs can be used simultaneously at ± 30 deg to both sides of the Frigate. In all other areas, only one STIR can be used. The CIWS has its own pointing radar which has a blind zone of ± 15 deg to the front of the ship. More details of the model for hardkill can be found in **?**.

The AWW softkill weapons use techniques to deceive or disorient a threat to cause the threat to destroy itself, or at least lose its fix on its intended victim. Again, the range and effectiveness of these weapons varies considerably. The AWW softkill weapons for a typical Frigate include chaff and jamming systems. The chaff system launches a shell that produces a burst at a designated position. The resultant chaff cloud has a significant radar cross-section that can be used to screen the Frigate or produce an alternate target on which a radar-guided threat can fix. The jamming system uses electromagnetic emissions to confuse the threats sensors to cause the threat to either lose its fix on its intended target, or to improperly assess the position of its target.

The exact nature of the specifications and capabilities of the various AWW hardkill and softkill weapons on real Frigates is obviously very complex, and much of that information is Classified. To avoid this issue, and in order to maintain emphasis on the research interests and not be burdened by the complexity and fidelity of the representation of hardkill and softkill, a considerably simplified model of the relevant AAW hardkill and softkill weapons was used. This model is a simple, non-classified version of AWW hardkill and softkill for a typical Frigate. The results could eventually be applied to the Canadian HALIFAX Class Frigate.

The softkill weapons system consists of two types of resources, Jamming and Chaff. In this application, there are two Jammers and four Chaff launch-

ers. Jammers can act on two threats each. During an attack, Jamming and Chaff must act concurrently and in a complementary way. First, Jamming is used to break the threat's radar lock on the Frigate. Once the missile has lost its target, Jamming creates a false target position on the threat's radar. Then Chaff is deployed at a position consistent with the false one provided by the Jammer. In this way, the threat's radar locks onto the chaff cloud as its new target.

2.1 Resource Allocation as a MDPs

Both the efficiency of harkill and softkill weapons are stochastic. In our problem, we assume 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:

- $Res = \langle res_1, ..., res_{|Res|} \rangle$ 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 $\langle Ta, \langle res_1, ..., res_{|Res_c|} \rangle \rangle$, 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}.
- A discount factor $\gamma \leq 1$.

A solution of an MDP is a policy π 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:

$$V^{\star}(s) = R(s) + \max_{a \in A(s)} \gamma \sum_{s' \in S} P_a(s'|s) V(s') \tag{1}$$

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. Furthermore, one may compute the Q-Values $Q(a, s)$ of each state action pair using the following equation:

$$Q(a, s) = R(s) + \gamma \sum_{s' \in S} P_a(s'|s) \max_{a' \in A(s')} Q(a', s') \tag{2}$$

where the optimal value of a state is $V^{\star}(s) = \max_{a \in A(s)} Q(a, s)$. The policy is subjected to the local resource constraints $res(\pi(s)) \leq L_{res} \forall\, s \in S$, 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 π. 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 $res(tra) \leq G_{res} \forall\ tra \in TRA$,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. Since resource allocation in a stochastic environment is NP-Complete, heuristics should be employed. In particular, heuristic search reduces the complexity of a stochastic resource allocation problem by focussing on relevant states, and by pruning the action space, but usually requires tight bounds. The next sections describe two separate methods to define $h_L(s)$ and $h_U(s)$ which are admissible heuristics for the lower bound value of a state $L(s)$ and upper bound value of a state $U(s)$. First of all, the method of Singh and Cohn [1998] is briefly described. Then, our own method proposes tighter bounds, thus allowing a more effective pruning of the action space.

2.2 Singh and Cohn's Bounds

Singh and Cohn [1998] defined lower and upper bounds to prune the action space in an MDP. 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 are as follows: $h_L(s) = \max_{ta \in Ta} V_{ta}(s_{ta})$, and $h_U(s) = \sum_{ta \in Ta} V_{ta}(s_{ta})$. 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. For readability, the upper bound by Singh and Cohn is named SinghU, and the lower bound is named SinghL.

2.3 Reducing the Upper Bound

SinghU includes actions which may not be possible to execute because of resource constraints, which overestimates the upper bound. To consider only possible actions, our upper bound, named maxU is introduced:

$$h_U(s) = \max_{a \in A(s)} \sum_{ta \in Ta} Q_{ta}(a_{ta}, s_{ta}) \tag{3}$$

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 Labelled RTDP (LRTDP) Bonet and Geffner [2003b] approach.

Theorem 1 *The upper bound defined by Equation 3 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 the MAXU bound in a state has a complexity of $\mathcal{O}(|A| \times |Ta|)$, and $\mathcal{O}(|Ta|)$ for SINGHU. A standard Bellman backup has a complexity of $\mathcal{O}(|A| \times |S|)$. 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 the upper bound is negligible.

2.4 Increasing the Lower Bound

The idea to increase SINGHL is to allocate the resources a priori among the tasks. 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 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:

$$mr_{ta}(s_{ta}) = \max_{a_{ta} \in A(s_{ta})} Q_{ta}(a_{ta}, s_{ta}) - \max_{a_{ta} \in A(s_{ta})} Q_{ta}(a_{ta} | res \notin a_{ta}, s_{ta}) \quad (4)$$

The concept of marginal revenue of a resource is used in Algorithm 2.1 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 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, in Line 9, 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 12 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 21, 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 23 with the highest marginal revenue, adjusted with residual value. In Line 24, the resource type res is allocated to the group of resources Res_{ta} of task ta. Afterwards, Line 29 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 $\frac{\max\limits_{a_{ta} \in A(s_{ta})} Q_{ta}(a_{ta}, s_{ta}(res))}{V_{ta}(s_{ta})}$, 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 32. When the global solution is computed, the lower bound is as follow:

$$h_L(s) = \max(\text{SINGHL}, \max_{a \in A(s)} \sum_{ta \in Ta} Low_{ta}(s_{ta})) \quad (5)$$

We use the maximum of the SINGHL bound and the sum of the lower bound components Low_{ta}, thus MARGINAL-REVENUE $\geq$ SINGHL. In particular, the SINGHL bound may be higher when a little number of task remains. 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 32 where a value for each task has to be computed with the shared resource. However, since the resource 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 2 *The lower bound of Equation 5 is admissible.*

Proof: $Low_{ta}(s_{ta})$ is computed with the resource 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 an admissible lower bound. ■

2.5 BRTDP and FRTDP

We now briefly describe Bounded RTDP (BRTDP) by McMahan *et al.* [2005] and Focused RTDP (FRTDP) by Smith and Simmons [2006]. Both these approaches 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 τ is a constant > 1. On the other hand, FRTDP stops the current trajectory using an adaptive maximum depth termination. The length D 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.

3 Experimental Results

3.1 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]. For the experiments we used the C++ code implemented by Smith and Simmons [2006]. We developped BRTDP in their simulator, which is initialized with the same (loose) initial lower and upper bounds Smith and Simmons used. Table 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.

We suppose 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 assume that the little improvement with pruning is due to the fact that the bounds are very loose and it requires

Table 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-3	large-b-w	large-ring-3	large-ring-w
FRTDP	0.49(7.14)	0.85(26.81)	0.37(7.46)	0.96(37.60)
BRTDP	**0.41**(5.94)	**0.74**(24.74)	0.37(7.58)	0.84(33.97)
P-FRTDP	0.50(6.95)	0.86(25.24)	0.37(**7.19**)	0.98(34.65)
P-BRTDP	**0.41 (5.72)**	**0.74 (23.37)**	**0.36**(7.21)	**0.80 (31.58)**

many backup for the pruning to start.

3.2 Resource Allocation

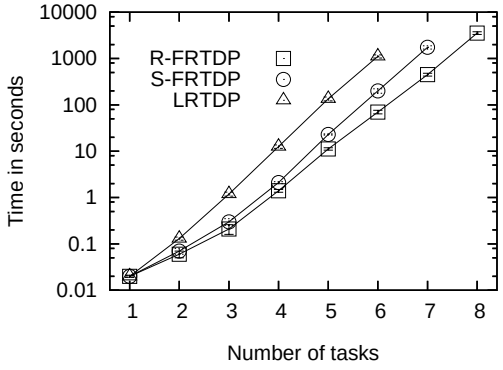

Figure 1: Computational efficiency of S-FRTDP, R-FRTDP and LRTDP.

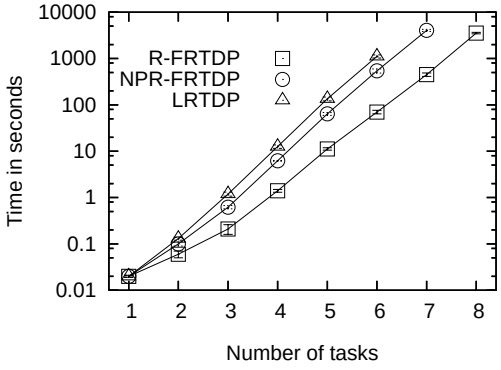

Figure 2: Computational efficiency of R-FRTDP, NPR-FRTDP and LRTDP.

We also tested BRTDP and FRTDP in a resource allocation problem. The domain is a the one described in Section 2. 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 τ 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 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. 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. The complexity of a resource allocation problem is more in the branching factor and the number of action in each state. In this case, 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 this table, and in Figures 1 and 2, 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 1112 seconds to plan for an LRTDP approach with six tasks (see Figure 1). The S-FRTDP approach diminished the planning time by using a lower and upper bound to prune the action space and with the efficient trajectories. M-FRTDP further reduce 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 spaces and produce efficient trajectories. For example, the planning time to obtain the optimal solution of R-FRTDP for problems with 4 tasks is 1.4 seconds, which is not excessive for our real-time resource allocation problem. Furthermore, as discussed by Singh and Cohn Singh and Cohn [1998], a two bounded heuristic search 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 choose to use another greedy algorithm of one's choice, which outputs a fast and near optimal solution. Furthermore, 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. Singh and Cohn 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.

On Figure 2, we may also observe that the action space pruning is much more efficient for our resource allocation problem than it is 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.

4 Conclusion

This paper has many interesting results. First of all, we implemented BRTDP 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 resource allocation problem, in the context of above-water warfare for a Frigate. In this problem, FRTDP converges faster than BRTDP, and we argue the low distance between the goal states with s_0 may explain this situation. Also, the initial bounds we proposed enables a faster convergence in comparison with the Singh and Cohn [1998] bounds. 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 is the case for hardkill and softkill weapons of a Frigate.

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.

References

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

A. Beynier and A. I. Mouaddib. An iterative algorithm for solving constrained decentralized markov decision processes. In *Proceeding of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, 2006.

B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, August 2003.

B. Bonet and H. Geffner. Labeled LRTDP approach: Improving the convergence of real-time dynamic programming. In *Proceeding of the Thirteenth*

International Conference on Automated Planning & Scheduling (ICAPS-03), pages 12–21, Trento, Italy, 2003.

E. A. Hansen and S. Zilberstein. LAO* : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML '05: Proceedings of the Twenty-Second International Conference on Machine learning*, pages 569–576, New York, NY, USA, 2005. ACM Press.

R. S. Pindyck and D. L. Rubinfeld. *Microeconomics*. Prentice Hall, 2000.

S. Singh and D. Cohn. How to dynamically merge markov decision processes. In *Advances in Neural Information Processing Systems*, volume 10, pages 1057–1063, Cambridge, MA, USA, 1998. MIT Press.

T. Smith and R. Simmons. 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)*, Boston, USA, 2006.

W. Zhang. Modeling and solving a resource allocation problem with soft constraint techniques. Technical report: WUCS-2002-13, Washington University, Saint-Louis, Missouri, 2002.

1 Annex

Algorithm 2.1 The marginal revenue lower bound algorithm.

Function REVENUE-BOUND(S)
returns a lower bound Low_{Ta}
for all $ta \in Ta$ **do**
  $V_{ta} \leftarrow$LRTDP(S_{ta})
  $value_{ta} \leftarrow 0$
end for
$s \leftarrow s_0$
repeat
  $res \leftarrow$ Select a resource type $res \in Res$
  for all $ta \in Ta$ **do**
    if res is consumable **then**
      $mr_{ta}(s_{ta}) \leftarrow V_{ta}(s_{ta}) - V_{ta}(s_{ta}(Res \setminus res))$
    else
      $mr_{ta}(s_{ta}) \leftarrow 0$
      repeat
        $mr_{ta}(s_{ta}) \leftarrow mr_{ta}(s_{ta}) + V_{ta}(s_{ta})$ - $\max_{(a_{ta} \in A(s_{ta}) | res \notin a_{ta})} Q_{ta}(a_{ta}, s_{ta})$
        $s_{ta} \leftarrow s_{ta}$.PICKNEXTSTATE(Res_c)
      until s_{ta} is a goal
      $s \leftarrow s_0$
    end if
    $mrrv_{ta}(s_{ta}) \leftarrow mr_{ta}(s_{ta}) \times \frac{V_{ta}(s_{ta}) - value_{ta}}{R(s_{g_{ta}})}$
  end for
  $ta \leftarrow$ Task $ta \in Ta$ which maximize $mrrv_{ta}(s_{ta})$
  $Res_{ta} \leftarrow Res_{ta} \bigcup \{res\}$
  $temp \leftarrow \emptyset$
  if res is consumable **then**
    $temp \leftarrow res$
  end if
  $value_{ta} \leftarrow value_{ta} + ((V_{ta}(s_{ta}) - value_{ta}) \times \frac{\max_{a_{ta} \in A(s_{ta}, res)} Q_{ta}(a_{ta}, s_{ta}(temp))}{V_{ta}(s_{ta})})$
until all resource types $res \in Res$ are assigned
for all $ta \in Ta$ **do**
  $Low_{ta} \leftarrow$LRTDP(S_{ta}, Res_{ta})
end for
return Low_{Ta}