A Q-decomposition and Bounded RTDP Approach to Resource Allocation

Pierrick Plamondon and Brahim Chaib-draa
Computer Science & Software Engineering Dept
Laval University
Québec, Canada
{plamon, chaib}@damas.ift.ulaval.ca

Abder Rezak Benaskeur
Decision Support Systems Section
Defence R&D Canada — Valcartier
Québec, Canada
abderrezak.benaskeur@drdc-rddc.gc.ca

ABSTRACT

This paper contributes to solve effectively stochastic resource allocation problems known to be NP-Complete. To address this complex resource management problem, a Q-decomposition approach is proposed when the resources which are already shared among the agents, but the actions made by an agent may influence the reward obtained by at least another agent. The Q-decomposition allows to coordinate these reward separated agents and thus permits to reduce the set of states and actions to consider. On the other hand, when the resources are available to all agents, no Q-decomposition is possible and we use heuristic search. In particular, the bounded Real-time Dynamic Programming (bounded RTDP) is used. Bounded RTDP concentrates the planning on significant states only and prunes the action space. The pruning is accomplished by proposing tight upper and lower bounds on the value function.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence.

General Terms

Algorithms, Performance, Experimentation.

Keywords

Heuristic Search, Q-decomposition, Marginal Revenue.

1. INTRODUCTION

This paper aims to contribute to solve complex stochastic resource allocation problems. In general, resource allocation problems are known to be NP-Complete [12]. 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.

There can be many types of resource allocation problems. Firstly, if the resources are already shared among the agents, 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 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. To solve this problem efficiently, we adapt Q-decomposition proposed by Russell and Zimdars [9]. In our Q-decomposition approach, a planning agent manages each task and all agents have to share the limited resources. The planning process starts with the initial state s_0. In s_0, each agent computes their respective Q-value. 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 agents. When implemented with heuristic search, since the number of states and actions to consider when computing the optimal policy is exponentially reduced compared to other known approaches, Q-decomposition allows to formulate the first optimal decomposed heuristic search algorithm in a stochastic environments.

On the other hand, when the resources are available to all agents, no Q-decomposition is possible. 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) [1], LRTDP [4], HDP [3], and LAO* [5] 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.* [1] 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 action space can be pruned of sub-optimal actions. To do this, McMahan *et*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.
Copyright 2007 IFAAMAS .

978-81-904262-7-5 (RPS) ©2007 IFAAMAS

al. [6], Smith and Simmons [11], and Singh and Cohn [10] proposed solving a stochastic problem using a RTDP type heuristic search with upper and lower bounds on the value of states. McMahan *et al.* [6] and Smith and Simmons [11] suggested, in particular, an efficient trajectory of state updates to further speed up the convergence, when given upper and lower bounds. This efficient trajectory of state updates can be combined to the approach proposed here since this paper focusses on the definition of tight bounds, and efficient state update for a constrained resource allocation problem.

On the other hand, the approach by Singh and Cohn is suitable to our case, and extended in this paper using, in particular, the concept of *marginal revenue* [7] to elaborate tight bounds. This paper proposes new algorithms to define upper and lower bounds in the context of a RTDP heuristic search approach. Our marginal revenue bounds are compared theoretically and empirically to the bounds proposed by Singh and Cohn. Also, even if the algorithm used to obtain the optimal policy is RTDP, our bounds can be used with any other algorithm to solve an MDP. The only condition on the use of our bounds is to be in the context of stochastic constrained resource allocation. The problem is now modelled.

2. PROBLEM FORMULATION

A simple resource allocation problem is one where there are the following two tasks to realize: $ta_1 = \{$wash the dishes$\}$, and $ta_2 = \{$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: $res_1 = \{$brush$\}$, and $res_2 = \{$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.

2.1 Resource Allocation as a MDPs

In our problem, 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}. 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 γ, which is a real number between 0 and 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:

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

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:

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

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. Q-decomposition which decomposes a planning problem to many agents to reduce the computational complexity associated to the state and/or action spaces is now introduced.

2.2 Q-decomposition for Resource Allocation

There can be many types of resource allocation problems. Firstly, if the resources are already shared among the agents, 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 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. 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 3, which is a naval platform which must counter incoming missiles (i.e. tasks) by using its resources (i.e. weapons, movements). 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 interaction 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.

To result these conflicts, we use Q-decomposition proposed by Russell and Zimdars [9] 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 learning 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 updated, an agent i considers the value as its respective contribution, or Q-value, 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 [8] 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:

$$Q_i(a_i, s_i) = R_i(s_i) + \gamma \sum_{s'_i \in S_i} P_{a_i}(s'_i|s_i) Q_i(a'_i, s'_i) \quad (3)$$

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 [9] demonstrated that local Sarsa converges to the optimum. Also, in some cases, this form of agent decomposition allows the local Q-functions to be expressed by a much reduced state and action space.

For our resource allocation problem described briefly 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 1. In Line 5 of the QDEC-BACKUP function, each agent managing a task computes its respective Q-value. Here, $Q^\star_i(a'_i, s')$ determines the optimal Q-value of agent i in state s'. An agent i uses as the value of a possible state transition s' the Q-value for this agent which determines the maximal global Q-value for state s' as in the original Q-decomposition approach. In brief, for each visited states $s \in S$, 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.

Then, the arbitrator functionalities are 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 allocate 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.

Algorithm 1 The Q-decomposition Bellman Backup.

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**

A standard Bellman backup has a complexity of $\mathcal{O}(|A| \times |S_{Ag}|)$, 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 $\mathcal{O}((|Ag| \times |A_i| \times |S_i|) + (|A| \times |Ag|))$, where $|S_i|$ is the number of states for an agent i, excluding the resources and $|A_i|$ is the 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 type 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.

However, when the resources are available to all agents, no Q-decomposition is possible. In this case, Bounded Real-Time Dynamic Programming (BOUNDED-RTDP) 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. BOUNDED-RTDP is now introduced.

2.3 Bounded-RTDP

Bonet and Geffner [4] proposed LRTDP as an improvement to RTDP [1]. LRTDP is a simple dynamic programming algorithm that involves a sequence of trial runs, each starting in the initial state s_0 and ending in a goal or a *solved* state. Each LRTDP trial is the result of simulating the policy π while updating the values $V(s)$ using a Bellman backup (Equation 1) over the states s that are visited. $h(s)$ is a heuristic which define 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 $V^\star(s)$ 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 for the deterministic version. 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 [4]. The convergence is accomplished by means of a labeling procedure called CHECKSOLVED(s, ϵ). This procedure tries to label as solved each traversed state in the current trajectory. When the initial state is labelled as solved, the algorithm has converged.

In this section, a bounded version of RTDP (BOUNDED-RTDP) is presented in Algorithm 2 to prune the action space of sub-optimal actions. This pruning enables to speed up the convergence of 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 CHECKSOLVED(s, ϵ) 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 [11] 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 $Q_U(a, s) \leq L(s)$ 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 ϵ. 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, PICKNEXTSTATE(res) 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 [11] [6].

As discussed by Singh and Cohn [10], this type of algorithm 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

Algorithm 2 The BOUNDED-RTDP algorithm. Adapted from [4] and [10].

Function BOUNDED-RTDP(S)
returns a value function V
repeat
  $s \leftarrow s_0$
  $visited \leftarrow null$
  repeat
    $visited.push(s)$
    BOUNDED-BACKUP(s)
    $Res_c \leftarrow Res_c \setminus \{\pi(s)\}$
    $s \leftarrow s$.PICKNEXTSTATE(Res_c)
  until s is a goal
  while $visited \neq null$ **do**
    $s \leftarrow visited.pop()$
    BOUNDED-BACKUP(s)
  end while
until s_0 is *solved* **or** $|A(s)| = 1 \; \forall \; s \in S$ reachable from s_0
return V

Algorithm 3 The bounded Bellman backup.

Function BOUNDED-BACKUP(s)
for all $a \in A(s)$ **do**
  $Q_U(a,s) \leftarrow R(s) + \gamma \sum_{s' \in S} P_a(s'|s) U(s')$
  $Q_L(a,s) \leftarrow R(s) + \gamma \sum_{s' \in S} P_a(s'|s) L(s')$
  {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}
  if $Q_U(a,s) \leq L(s)$ **then**
    $A(s) \leftarrow A(s) \setminus res(a)$
  end if
end for
$L(s) \leftarrow \max_{a \in A(s)} Q_L(a,s)$
$U(s) \leftarrow \max_{a \in A(s)} Q_U(a,s)$
$\pi(s) \leftarrow \arg\max_{a \in A(s)} Q_L(a,s)$
if $|U(s) - L(s)| < \epsilon$ **then**
  $s \leftarrow solved$
end if

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 [10] proved that an algorithm that uses admissible lower and upper bounds to prune the action space is assured of converging to an optimal solution.

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 [10] is briefly described. Then, our own method proposes tighter bounds, thus allowing a more effective pruning of the action space.

2.4 Singh and Cohn's Bounds

Singh and Cohn [10] 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, $h_L(s) = \max_{ta \in Ta} V_{ta}(s_{ta})$, 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. To determine the optimal policy π, Singh and Cohn implemented an algorithm very similar to BOUNDED-RTDP, which uses the bounds to initialize $L(s)$ and $U(s)$. The only difference between BOUNDED-RTDP, and the RTDP version of Singh and Cohn is in the stopping criteria. 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 ϵ. 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 [11] and McMahan *et al.* BRTDP [6]. In this paper, 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.

2.5 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:

$$h_U(s) = \max_{a \in A(s)} \sum_{ta \in Ta} Q_{ta}(a_{ta}, s_{ta}) \quad (4)$$

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 2.1. *The upper bound defined by Equation 4 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.

2.6 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 [7], and has recently been used for coordination of a Decentralized MDP [2]. 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:

$$mr_{ta}(s_{ta}) = \max_{a_{ta} \in A(s_{ta})} Q_{ta}(a_{ta}, s_{ta}) - \max_{a_{ta} \in A(s_{ta})} Q_{ta}(a_{ta} | res \notin a_{ta}, s_{ta}) \quad (5)$$

The concept of marginal revenue of a resource is used in Algorithm 4 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 [4] 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 $\frac{\max_{a_{ta} \in A(s_{ta})} Q_{ta}(a_{ta}, s_{ta}(res))}{V_{ta}(s_{ta})}$, which is the ratio of the efficiency of resource type res. In other words, $value_{ta}$ is assigned to $value_{ta}$ + (*the residual value* × *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:

$$h_L(s) = \max(\text{SINGHL}, \max_{a \in A(s)} \sum_{ta \in Ta} Low_{ta}(s_{ta})) \quad (6)$$

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

Algorithm 4 The marginal revenue lower bound algorithm.

```
1: Function REVENUE-BOUND(S)
2: returns a lower bound Low_Ta
3: for all ta ∈ Ta do
4:    V_ta ←LRTDP(S_ta)
5:    value_ta ← 0
6: end for
7: s ← s_0
8: repeat
9:    res ← Select a resource type res ∈ Res
10:   for all ta ∈ Ta do
11:      if res is consumable then
12:         mr_ta(s_ta) ← V_ta(s_ta) − V_ta(s_ta(Res \ res))
13:      else
14:         mr_ta(s_ta) ← 0
15:         repeat
16:            mr_ta(s_ta) ← mr_ta(s_ta) + V_ta(s_ta) -
                  max_{(a_ta ∈ A(s_ta) | res ∉ a_ta)} Q_ta(a_ta, s_ta)
17:            s_ta ← s_ta.PICKNEXTSTATE(Res_c)
18:         until s_ta is a goal
19:         s ← s_0
20:      end if
21:      mrrv_ta(s_ta) ← mr_ta(s_ta) × (V_ta(s_ta) − value_ta) / R(s_{g_ta})
22:   end for
23:   ta ← Task ta ∈ Ta which maximize mrrv_ta(s_ta)
24:   Res_ta ← Res_ta ⋃ {res}
25:   temp ← ∅
26:   if res is consumable then
27:      temp ← res
28:   end if
29:   value_ta ← value_ta + ((V_ta(s_ta) − value_ta) ×
            max_{a_ta ∈ A(s_ta, res)} Q_ta(a_ta, s_ta(temp)) / V_ta(s_ta))
30: until all resource types res ∈ Res are assigned
31: for all ta ∈ Ta do
32:    Low_ta ←LRTDP(S_ta, Res_ta)
33: end for
34: return Low_Ta
```

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 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 2.2. *The lower bound of Equation 6 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. ■

3. DISCUSSION AND EXPERIMENTS

The domain of the experiments 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. The number of resource type has been fixed to 5, where there are 3 consumable resource types and 2 non-consumable resources types.

For this problem a standard lrtdp approach has been implemented. 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. The approaches described in this paper are compared in Figures 1 and 2. Lets summarize these approaches here:

- Qdec-lrtdp: The backups are computed using the Qdec-backup function (Algorithm 1), but in a lrtdp context. In particular the updates made in the check-Solved function are also made using the the Qdec-backup function.
- lrtdp-up: The upper bound of maxU is used for lrtdp.
- Singh-rtdp: The SinghL and SinghU bounds are used for bounded-rtdp.
- mr-rtdp: The revenue-bound and maxU bounds are used for bounded-rtdp.

To implement Qdec-lrtdp, we divided the set of tasks in two equal parts. The set of task Ta_i, managed by agent i, can be accomplished with the set of resources Res_i, while the second set of task $Ta_{i'}$, managed by agent $Ag_{i'}$, can be accomplished with the set of resources $Res_{i'}$. Res_i had one consumable resource 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 was assigned to $Ta_{i'}$. There are constraint between the group of resource Res_i and $Res_{i'}$ such that some assignments are not possible. These constraints are managed by the arbitrator as described in Section 2.2. Q-decomposition permits to diminish the planning time significantly in our problem settings, and seems a very efficient approach when a group of agents have to allocate resources which are only available to themselves, but the actions made by an agent may influence the reward obtained by at least another agent.

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, notice that the lrtdp lrtdp-up and approaches for resource allocation, which doe 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 1). The Singh-rtdp approach diminished the planning time by using a lower and upper bound to prune the action space. mr-rtdp further reduce the planning time by providing very tight initial bounds. In particular, Singh-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 mr-rtdp with the SinghU bound, and this was slightly less efficient than with the maxU bound. We also implemented mr-rtdp with the SinghL bound, and this was slightly more efficient than Singh-rtdp. From these results, we conclude that the difference of efficiency between mr-rtdp and Singh-rtdp is more attributable to the marginal-revenue lower bound that to the maxU upper bound. Indeed, when the number of task to execute is high, the lower bounds by Singh-rtdp takes the values of a single task. On the other hand, the lower bound of mr-rtdp takes into account the value of all

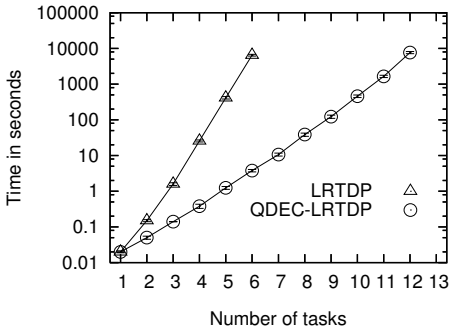

Figure 1: Efficiency of Q-decomposition LRTDP and LRTDP.

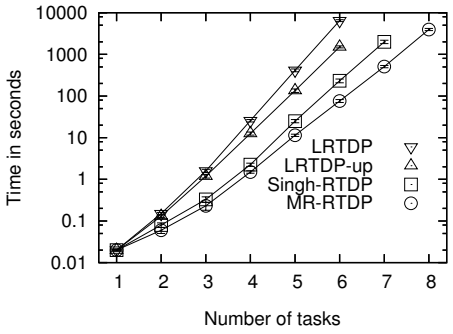

Figure 2: Efficiency of MR-RTDP compared to SINGH-RTDP.

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.

4. CONCLUSION

The experiments have shown that Q-decomposition seems a very efficient approach when a group of agents have to allocate resources which are only available to themselves, but the actions made by an agent may influence the reward obtained by at least another agent.

On the other hand, when the available resource are shared, no Q-decomposition is possible and we proposed tight bounds for heuristic search. In this case, the planning time of BOUNDED-RTDP, which prunes the action space, is significantly lower than for LRTDP. Furthermore, The marginal revenue bound proposed in this paper compares favorably to the Singh and Cohn [10] approach. BOUNDED-RTDP with our proposed bounds may apply to a wide range of stochastic environments. The only condition for the use our bounds is that each task possesses consumable and/or non-consumable limited resources.

An interesting research avenue would be to experiment our bounds with other heuristic search algorithms. For instance, FRTDP [11], and BRTDP [6] are both efficient heuristic search algorithms. In particular, both these approaches proposed an efficient state trajectory updates, when given upper and lower bounds. Our tight bounds would enable, for both FRTDP and BRTDP, to reduce the number of backup to perform before convergence. Finally, the BOUNDED-RTDP function prunes the action space when $Q_U(a, s) \leq L(s)$, as Singh and Cohn [10] suggested. FRTDP and BRTDP could also prune the action space in these circumstances to further reduce their planning time.

5. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical report CUED/FINFENG/TR 166, Cambridge University Engineering Department, 1994.

[9] S. J. Russell and A. Zimdars. Q-decomposition for reinforcement learning agents. In *ICML*, pages 656–663, 2003.

[10] 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.

[11] 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.

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