A Real-time Dynamic Programming Decomposition Approach to Resource Allocation

Pierrick Plamondon [1#], Brahim Chaib-draa [1], Abder Rezak Benaskeur [2]
[1] *Computer Science Department, Laval University*
Québec, PQ, G1K 7P4, Canada, {plamon; chaib}@damas.ift.ulaval.ca
[2] *Decision Support Systems Section, Defence R&D Canada — Valcartier*
Québec, PQ, G3J 1X5, 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, the merging of two approaches is made: The Q-decomposition model, which coordinates reward separated agents through an arbitrator, and the Labeled Real-Time Dynamic Programming (*LRTDP*) approaches are adapted in an effective way. The Q-decomposition permits to reduce the set of states to consider, while* LRTDP *concentrates the planning on significant states only. As demonstrated by the experiments, combining these two distinct approaches permits to further reduce the planning time to obtain the optimal solution of a resource allocation problem.*

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 [10]. 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 problem of large MDPs is based on problem decomposition. For instance, Markov task decomposition [8], and Q-decomposition [7] are all decomposition techniques, where the original MDP is decomposed in sub-MDPs such that the search is focussed on a particular region at a time. A heuristic to combine the sub-MDPs is generally used to improve the value of the resulting policy, or solution. When decomposing the state space, the policy is a (perhaps sub-optimal) solution to the original global MDP.

Another interesting approach is that of heuristic search where many algorithms have been developed recently. The idea of heuristic search is to start from the initial state, and given an admissible heuristic for the value of unvisited states, a huge part of the state space may be omitted from the search. For instance RTDP [2], LRTDP [5], HDP [6], and LAO* [1] are all state of the art heuristic search approaches used in a stochastic environment. Because of its anytime flavor, an interesting approach is Real-Time Dynamic Programming (RTDP) introduced by Barto et al. [2] which updates states in trajectories from an initial state s_0 to a goal state s_g in a very efficient manner. Then, Bonet and Geffner [5] proposed a labelling procedure to accelerate the convergence of RTDP in their L(Labeled)RTDP algorithm.

In this paper, a merging of the Q-decomposition adapted from Russell and Zimdars [7] with the LRTDP approach is presented. The original Q-decomposition reinforcement learning coordination process determines the Q-values which maximize the sum of all combinations of possible agent Q-value for each visited state. An important assumption of this method is that each agent should have its own separate reward function. As a consequence, for a resource allocation problem, there would be an agent for each task to achieve.

In our Q-decomposition LRTDP approach (QDEC-LRTDP), 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-values. Then, the planning agents are coordinated through an arbitrator to find the highest global Q-value by adding the respective possible Q-values. The action which determines the maximal Q-value is executed in the environment to determine the next state, and so on until a goal state is reached. These trajectories are updated until all reachable states under the current policy are solved. Notice that our QDEC-LRTDP is guaranteed to converge to an optimal policy. Since the number of possible state transitions to consider when computing the optimal policy is greatly reduced compared to LRTDP, QDEC-LRTDP appears to be the first optimal decomposed heuristic search algorithm in a stochastic environments. Indeed, although our work focusses on resource bounded agents, QDEC-LRTDP may apply to a wide range of stochastic environments. The only condition for the use of QDEC-LRTDP is that each agent should have its own separate reward function. QDEC-LRTDP is explained in more detail in Section 2-D. For now, the problem which motivated this paper is now described and modelled.

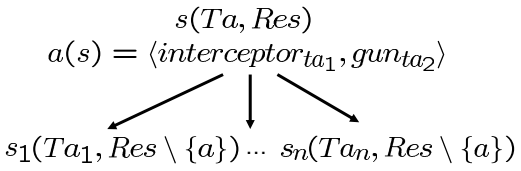

Fig. 1: State transition graph.

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 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.

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.

Specifically, the resources may be constrained by the number that may be used at a time (local constraint), and in total (global constraint). 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. 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.

A state s includes the joint states of the threats (tasks) to counter, and the available resources. Figure 1 describes the state transition process. The system is in a state s with a set of task Ta to realize, and a set of resource Res available. A possible action in this state may be to allocate one missile interceptor to threat ta_1, and the gun to task ta_2. The system changes stochastically of state, as the state of the tasks may change. For example, a threat may be destroyed or not with a certain probability, after having allocated theses resources. In this example, the state of the task may change in n new possible combinations. For all these n possible state transitions after s, the consumable resources available are $Res_c \setminus res(a)$, where $res(a)$ are the consumable resources used by action a.

A. Markov Decision Processes (MDPs) in the Context of Resource Allocation

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 represents the particular state s_{ta}, which is the characteristic of each unaccomplished task $ta \in Ta$ in the environment, and the available consumable resources. 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 there 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 is given when the state of a task (s_{ta}) is in an achieved state, and 0 in all other cases.
- A discount factor γ, 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 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 $Res_c \backslash res(a)$, 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 $\{\pi(s)\} \leq L_{res} \forall\ s \in S$, and $\forall\ res \in Res$. The global constraint for consumable resources 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 mapping 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. 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.

B. Q-decomposition for Resource Allocation

Russell and Zimdars [7] proposed Q-decomposition 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 [9] 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 $Res_c \setminus res(a)$. Russell and Zimdars [7] 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. It is now determined whether Q-decomposition allows reducing the state and/or action space for the resource allocation problem described in Section 2.

In the context of allocating resources to tasks, the rewards are obtained when each particular task is achieved. Thus, to use Q-decomposition, an agent would have available the shared limited resources to realize each specific task. In other words, an agent would manage each task. There are two conditions which permit an efficient task decomposition as enumerated by Meuleau et al. [8]. First, the problem has to be composed of multiple tasks whose utilities are independent. Second, the actions taken (or resources allocated) with respect to a task cannot influence the status of any other task. If these two conditions are met, each task may therefore be viewed as an MDP. In this case, the state space may be decomposed in the state transition. Indeed, since each task has its own independent reward function, the tasks have no reason to be considered as a whole in a possible state transition for a Bellman update. Many applications respect these conditions. For example, job shop scheduling, sensor management, transport routing, weapon allocation, and so on are applications which fill these two criteria.

It has been determined that the state space may be decomposed for each task, but the action space has to be taken as a whole. Indeed, if the action space is decomposed, the possible state transition s'_i from state s used in a Bellman backup would consider $Res_c \setminus res(a_i)$ as the consumable resources remaining for agent i. It is likely that more than $res(a_i)$ resource is going to be used in state s. Indeed, the other agents are very likely to allocate resources to their current tasks in state s. This calculation of resource remaining would overestimate the value of state s. For this reason, the action space may not be decomposed for a constrained resource allocation problem if an optimal solution is desired. Heuristic search may also reduce the complexity of a stochastic resource allocation problem by focussing on relevant states. To this end, the Labeled Real-Time Dynamic Programming (LRTDP) heuristic search algorithm is now introduced.

C. LRTDP

Bonet and Geffner [5] proposed LRTDP as an improvement to RTDP [2]. 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 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 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. The convergence is accomplished by means of a labelling 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. The next section describes how to merge Q-decomposition and LRTDP to further reduce the planning time of a constrained resource allocation problem while guaranteeing optimality.

D. A Q-decomposition LRTDP Approach to Resource Allocation

We now show how to merge efficiently Q-decomposition and LRTDP to produce an optimal decomposition heuristic search algorithm. The method is presented in Algorithm 1, 2 and 3. The modifications from the original LRTDP algorithm are as follows. The value of a state s in Line 8 of the QDEC-LRTDP function is updated according to Algorithm 2. In Line 6 of the QDEC-LRTDP-BACKUP function, each agent managing a task computes its respective Q-value. Here, $Q_i^\star(a', s')$ 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 for an agent is the same as the state space for LRTDP. The gain in complexity to use Q-decomposition resides in the $\sum_{s'_i \in S_i} P_{a_i}(s'_i|s,a)$ part of the equation. An agent considers as a possible state transition only the possible states of the task it manages. Since the number of states is exponential with the number of tasks, using Q-decomposition should reduce the planning time significantly. Also, the state transition P_{a_i} for an agent considers the global action a because it is possible that the action executed on another task ta' influence the status of task ta managed by agent i. Thus, the condition of task decomposition cited by Meuleau et al. [8], which says that the actions taken (or resources allocated) with respect to a task cannot influence the status of any other task is respected since the action space is taken as a whole in QDEC-LRTDP. The global Q-value is the sum of the Q-values produced by each agent managing each task as shown in Line 7. Lines 9 to 11 simply allocate the current value and policy with respect to the highest global Q-value, as for LRTDP. In Line 13, the optimal Q-value of each agent is updated to the specific Q-value of each agent for action a.

Afterwards, when the execution is back to the QDEC-LRTDP function, to update both the task and resource parts of state s in Line 9 and 10. First of all, the new available set of resources is $Res_c \setminus \{\pi(s)\}$. Indeed, even if an agent i plans only for its task with the part a_i of the global action a, it has to consider the resources used by the other agents. The less the number of resources is available, the less the value of a state is expected to be, so the global resource consumption has to be taken into account. Then, the new set of tasks to accomplish is produced in Line 10. In brief, the PICKNEXTSTATE(Res_c) function randomly picks a none-$solved$ state, containing a new set of tasks to realize, by executing the current policy.

Algorithm 1 The Q-decomposition LRTDP algorithm.

```
1:  Function QDEC-LRTDP(S)
2:  returns a value function V
3:  repeat
4:    s ← s_0
5:    visited ← null
6:    repeat
7:      visited.PUSH(s)
8:      QDEC-LRTDP-BACKUP(s)
9:      Res_c ← Res_c \ {π(s)}
10:     s ← s.PICKNEXTSTATE(Res_c)
11:   until s is a goal
12:   while visited ≠ null do
13:     s ← visited.POP()
14:     if ¬ QDECCHECKSOLVED(s, ε) then
15:       break
16:     end if
17:   end while
18: until s_0 is solved
19: return V
```

The QDECCHECKSOLVED function (algorithm 3) is called in Line 14 of the QDEC-LRTDP function. This algorithm is very similar to the original CHECKSOLVED in LRTDP. The only difference is in Line 30. The original CHECKSOLVED function used to perform a standard Bellman Backup; on the other hand, the QDECCHECKSOLVED function calls the QDEC-LRTDP-BACKUP function.

Beside these changes from the original LRTDP, QDEC-LRTDP has the same stopping criteria as LRTDP. As a matter of fact, it has the same *expectation* of producing the same behavior as LRTDP. Here, the term expectation is used because the behavior of LRTDP is probabilistic since the next state to compute is produced using a probability distribution.

The behavior of QDEC-LRTDP is now discussed and we start it by proving the optimality of QDEC-LRTDP.This proof requires two steps. First of all, the fact that the value of a state in a given trajectory for QDEC-LRTDP is updated in the same manner as with the original LRTDP is shown. Then, the convergence of QDEC-LRTDP is proved.

Algorithm 2 The QDEC-LRTDP Bellman Backup.

Function QDEC-LRTDP-BACKUP(s)
$V(s) \leftarrow 0$
for all $a \in A(s)$ **do**
  $Q(a,s) \leftarrow 0$
  for all $i \in Ag$ **do**
    $Q_i(a,s) \leftarrow R_i(s) + \gamma \sum_{s'_i \in S_i} P_{a_i}(s'_i|s,a) Q^\star_i(a',s')$
    {where $Q^\star_i(a',s') = h_i(s')$ when s' is not yet visited, and s' has $Res_c \setminus res(a)$ remaining consumable resources}
    $Q(a,s) \leftarrow Q(a,s) + Q_i(a,s)$
  end for
  if $Q(a,s) > V(s)$ **then**
    $V(s) \leftarrow Q(a,s)$
    $\pi(s) \leftarrow a$
    for all $i \in Ag$ **do**
      $Q^\star_i(a,s) \leftarrow Q_i(a,s)$
    end for
  end if
end for

Lemma 2.1: A state in a trajectory for QDEC-LRTDP is updated in the same manner as for LRTDP.

Proof: The following equation is used in QDEC-LRTDP to compute the value of a state:

$$V(s) = \sum_{i \in Ag} R_i(s) + \max_{a \in A(s)} \gamma \sum_{s'_i \in S'_i} P_{a_i}(s'_i|s,a) Q^\star_i(a',s') \tag{4}$$

Since the reward can be additively decomposed for each task, Equation 4 may be rewritten as:

$$V(s) = R(s) + \max_{a \in A(s)} \sum_{i \in Ag} \gamma \sum_{s'_i \in S'_i} P_{a_i}(s'_i|s,a) Q^\star_i(a',s') \tag{5}$$

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 5 may be rewritten as:

$$V(s) = R(s) + \max_{a \in A(s)} \gamma \sum_{s' \in S'} P_a(s'|s) Q(a',s') \tag{6}$$

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 is the same as a Bellman backup, and $Q(a',s')$ is the same as $V(s)$, a state in a trajectory for QDEC-LRTDP is updated in the same manner as for LRTDP. ■

Lemma 2.2: The QDEC-LRTDP algorithm converges.

Proof: It has been proven that the value of a state is updated similarly as in the centralized LRTDP algorithm. Also, the newly visited state s' from the current state s is produced using the global action a, as for LRTDP. Since the states have the same value and the state transition has the same behavior as LRTDP, the convergence properties of QDEC-LRTDP are the same as the one of LRTDP. ■

Algorithm 3 The QDECCHECKSOLVED algorithm for QDEC-LRTDP.

Function QDECCHECKSOLVED(s, ϵ)
returns a boolean rv
$rv \leftarrow true$
$open \leftarrow$EMPTYSTACK
$closed \leftarrow$EMPTYSTACK
if $\neg s$.SOLVED **then**
  $open$.PUSH(s)
end if
while $open \neq$ EMPTYSTACK **do**
  $s \leftarrow open$.POP()
  $closed$.PUSH(s)
  {check residual}
  if $\delta(s) > \epsilon$ **then**
    $rv \leftarrow false$
    continue
  end if
  for all s' such that $P_{\pi(s)}(s'|s) > 0$ **do**
    if $\neg s$.SOLVED $\wedge \neg$ IN($s', open \cup closed$) **then**
      $open$.PUSH(s)
    end if
  end for
end while
if $rv = true$ **then**
  {label relevant states}
  for all $s' \in closed$ **do**
    s'.SOLVED $= true$
  end for
else
  while $closed \neq$ EMPTYSTACK **do**
    $s \leftarrow closed$.POP()
    QDEC-LRTDP-BACKUP(s)
  end while
end if
return rv

Theorem 1: The QDEC-LRTDP algorithm is optimal.

Proof: With lemma 2.1 and 2.2 being true, the result follows. Indeed, the initial s which is updated optimally, is guaranteed to converge in the same manner as LRTDP. ■

E. Complexity of Q-Decomposition

We first compare the complexity of the Q-decomposition part of MRQDEC-RTDP. A Bellman backup iteration LRTDP has a complexity of $\mathcal{O}(|A| \times |S_{Ta}|)$, where $|S_{Ta}|$ is the number of joint states for the tasks excluding the resources, and $|A|$ is the number of joint actions. In the MRQDEC-RTDP approach, a backup has a complexity of $\mathcal{O}(|Ta| \times |A| \times |S_{ta}|)$, where $|S_{ta}|$ is the number of states for each task, excluding the resources.

Since $|S_{Ta}|$ is combinatorial with the number of tasks, so $|S_{ta}| \ll |S|$. Furthermore, the communication cost between the agents and the arbitrator is null since this approach does not consider a geographically separated problem. QDEC-LRTDP and LRTDP are compared and discussed in the next section.

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), as described in Section 2. The different resources have their efficiency modified when used in conjunction on a same task, thus producing positive and negative interactions between resources. For the experiments, 100 randomly resource allocation problems were generated for all combinations of numbers of tasks and resource types, where one agent manages each task. 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.

For this problem, an admissible heuristic has to be formulated. 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 of overestimating its real value since the value of achieving a task ta is the highest the planner may get for ta. No other heuristic has been implemented since the objective here is only to compare QDEC-LRTDP and LRTDP in the same settings. However, even when using this simple heuristic, a huge part of the state space is still not visited when computing the optimal policy.

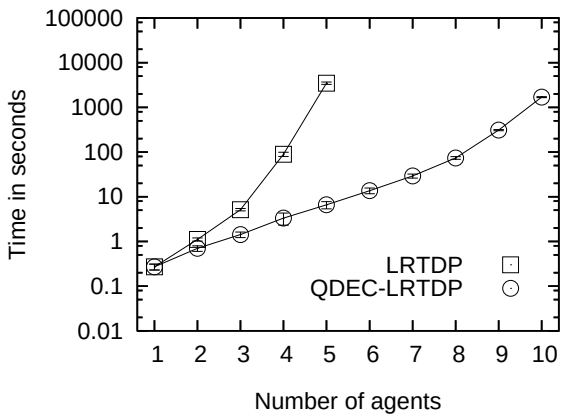

Fig. 2: Computational efficiency of LRTDP and QDEC-LRTDP.

The QDEC-LRTDP and LRTDP approaches are compared in Figure 2. In terms of experiments, notice that the LRTDP approach for resource allocation, which is computed on the joint action and state spaces of all agents, is much more complex. For instance, it takes an average of 3487 seconds to plan for an LRTDP approach with five tasks (see Figure 2). The QDEC-LRTDP approach solves optimally the same type of problem in an average of 6.58 seconds. Indeed, the time reduction is quite significant compared to LRTDP, which demonstrates the efficiency of QDEC-LRTDP to decompose the state space.

4. CONCLUSION

The experiments have shown that QDEC-LRTDP 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 paper has focussed 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 [6], and LAO* [1] are both efficient heuristic search algorithms and may be greatly improved if combined with Q-decomposition.

Other future work of QDEC-LRTDP would be to reduce the action space, which is the same as for LRTDP. In particular, the action space can be reduced using a lower and higher bound on the value of states [3] [4]. Using these bounds, if an action has its upper bound lower than the lower bound of a state, it can be pruned from the action space of QDEC-LRTDP for this state.

REFERENCES

[1] E. A. Hansen and S. Zilberstein "LAO* : A heuristic search algorithm that finds solutions with loops", *Artificial Intelligence*, vol. 129, no. 1-2, pp. 35-62, 2001.

[2] A. G. Barto and S. J. Bradtke and S. P. Singh, "Learning to act using real-time dynamic programming", *Artificial Intelligence*, vol. 72, no. 1, pp. 81-138, 1995.

[3] S. Singh and D. Cohn, "How to dynamically merge Markov decision processes", in *Proc. Advances in neural information processing systems*, Cambridge, MA, USA, 1998, pp. 1057-1063, MIT Press

[4] H. B. McMahan and M. L. and G. J. Gordon, "Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees", in *Proc. of the 22nd international conference on Machine learning*, New York, NY, USA, 2005, pp. 569-576, ACM Press

[5] B. Bonet and H. Geffner, "Labeled LRTDP approach: Improving the Convergence of Real-Time Dynamic Programming", in *Proc. of the 13Th International Conference on Automated Planning & Scheduling (ICAPS-03)*, Trento, Italy, 2003

[6] B. Bonet and H. Geffner, "Faster Heuristic Search Algorithms for Planning with Uncertainty and Full Feedback", in *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003

[7] S. J. Russell and A. Zimdars, "Q-Decomposition for Reinforcement Learning Agents", in *Proc. of the 20th international conference on Machine learning*, 2003, pp. 656-663

[8] N. Meuleau and M. Hauskrecht and K. Kim and L. Peshkin and L. P. Kaebling and T. Dean and C. Boutilier, "Solving very large weakly coupled Markov decision processes", in *Proc. of the 15th National Conference on Artificial Intelligence*, 1998, pp. 165-172

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

[10] W. Zhang, "Modeling and Solving a Resource Allocation Problem with Soft Constraint Techniques". Technical Report: WUCS-2002-13, Saint-Louis, Missouri, 2002