Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Dynamic Resource Routing using Real-Time Dynamic Programming

Sebastian Schmoll, Matthias Schubert
Ludwig-Maximilians-Universität, Munich, Germany
{schmoll, schubert}@dbs.ifi.lmu.de

Abstract

Acquiring available resources in stochastic environments becomes more and more important to future mobility. For instance, cities like Melbourne, Canberra and San Francisco install sensors that detect in real-time whether a parking spot (resource) is available or not. In such environments, the current state of the resources may be fully observable, although the future development is stochastic. In order to reduce the traffic, such cities want to fully exploit parking spots, such that the amount of searching cars is minimized. Thus, we formulate a problem setting where the expected seek time for each driver is minimized. This problem can be modeled by a Markov Decision Process (MDP) and solved using standard algorithms. In this paper, we focus on the setting, where pre-computation is not possible and search policies have to be computed on the fly. Our approach is based on state-of-the-art Real-Time Dynamic Programming (RTDP) approaches. However, standard RTDP approaches do not perform well on this specific problem setting as shown in our experiments. We introduce adapted bounds and approximations that exploit the specific nature of the problem in order to improve the performance significantly.

1 Introduction

The ongoing digitization opens up new applications for advanced planing algorithms that empower driving assistant systems and autonomous vehicles. For example, cities like Melbourne, Canberra and San Francisco have equipped on-street parking spots with sensors providing real-time information about their availabilities. This information can be used to guide a driver from his/her current position to an available spot. Similar tasks occur for charging stations for electric vehicles and drop-off locations for shared bicycles which could also provide real-time information about their availability. Finding a good search strategy is often a challenging task (c.f. [Jossé *et al.*, 2015]) because available resources might be sparse and change their state frequently. Thus, an available resource might become occupied while travelling to its location. To minimize search times, a search strategy should consider the provided real-time information and react dynamically to recent developments. We denote this problem as the dynamic resource routing problem (DRRP) and model it as Markov Decision Process under complete observation. Though, computing an optimal policy can be approached by standard algorithms, the state space of the problem grows exponentially with the number of resources and thus, the corresponding runtimes quickly become infeasible. We propose to compute policies for a given start location and a set of feasible resources on the fly. Though this requires interactive computation times, it also allows for the application of stochastic shortest path algorithms like real-time dynamic programming (RTDP), where only the subset of relevant states will be touched during the computation. However, even the state-of-the-art in general MDP computation [McMahan *et al.*, 2005] does not cope well with the complexity of the problem, we exploit the specific nature of the resource routing problem to propose a dedicated RTDP algorithm. Our new algorithm employs specialized lower and upper bounds on the utility. Additionally, we propose a new approximative technique to limit the number of considered state transitions to further improve the run-time efficiency. In our evaluation, we demonstrate that our new method outperforms state-of-the-art solutions for solving general MDPs on the road network of Melbourne where public data on the parking spot vacancy is available.

2 Related Work

The problem of finding an available resource in a spatial environment has been studied in [Jossé *et al.*, 2015; Guo and Wolfson, 2016]. Both methods assume that the exact current state of a resource (available or occupied) is unknown. [Jossé *et al.*, 2015] uses a branch and bound approach to compute a path (with loops) in a finite horizon setting. They introduce a time-continuous Markov chain model that models the resources' probability for being available, which we will use for our solution as well. [Guo and Wolfson, 2016] model the problem basically as Markov Decision Process and solve it in a finite and infinite horizon environment by methods which is similar to backward induction and Value Iteration (VI) [Bellman, 1957]. For more details on MDPs, we refer to [Puterman, 1994]. The first to model the resources' state in the MDP state are [Schmoll and Schubert, 2018]. They assume that there exists environments where the actual state of the parking spot is available and they show

with real-world datasets that modeling this information improves the parking search significantly. However, they assume that pre-computation of small spatial areas is possible and the user can choose a suitable area from the database. Thus, they propose to solve the MDP with VI. In contrast, in our work we focus on a setting where pre-computation is not possible. However, due to the huge state space, VI is not practical for on demand customer queries. [Bertsekas and Tsitsiklis, 1991] introduced a sub-problem of MDPs, called Stochastic Shortest Path, which can be solved by different promising approaches like LAO* [Hansen and Zilberstein, 2001] (an and-or tree heuristic search that allows loops) and Real-Time Dynamic Programming (RTDP), which was first introduced by [Barto *et al.*, 1995] and has been extended by [Bonet and Geffner, 2003b; 2003a; Smith and Simmons, 2006; McMahan *et al.*, 2005; Sanner *et al.*, 2009]. However, (improved) LAO* has been shown to be outperformed by Labeled-RTDP (LRTDP) [Bonet and Geffner, 2003b], an RTDP extension. LRTDP has been shown to be outperformed by Focused-RTDP and Bounded-RTDP (BRTDP) [Smith and Simmons, 2006; McMahan *et al.*, 2005]. Bayesian-RTDP seems to outperform BRTDP in some problem settings [Sanner *et al.*, 2009], nonetheless, in our internal experiments Bayesian-RTDP did not work very well with our problem setting due to the myopic Value of Perfect Information (VPI) computation. We focus on BRTDP in this paper even if our improvements could be adapted to Bayesian-RTDP as well.

3 Dynamic Resource Routing

The dynamic resource routing problem (DRRP) aims at finding available resources P that are spatially located in a Graph $G = (N, E, W)$, where N is the set of nodes, $E \subseteq N \times N$ is the set of edges in the graph and $W : E \to \mathbb{R}^+$ is the function that defines the expected costs of traversing one edge in E. Each resources may change from available to occupied and vice versa in an uncertain manner. Although it may be uncertain, we expect a given transition probability $\mathcal{X}^t_{f,f'} \in [0, 1]$ for each resource, whereas f is a boolean set $\{a, o\}$ (available/occupied) and $t \in \mathbb{R}^+$ defines time passed. One way of implementing such a transition probability function is by using Markov chains. We model the transition probabilities as continuous Markov chains as proposed in [Jossé *et al.*, 2015].

$$\mathcal{X}^t_{f,a}(p) = \begin{cases} \frac{\mu_p}{\lambda_p+\mu_p} + \frac{\lambda_p}{\lambda_p+\mu_p} e^{-(\lambda_p+\mu_p)t}, & \text{if } f = a \\ \frac{\mu_p}{\lambda_p+\mu_p} - \frac{\mu_p}{\lambda_p+\mu_p} e^{-(\lambda_p+\mu_p)t}, & \text{else} \end{cases}$$

μ_p and λ_p describe the multiplicative inverse of the average time the resource p becomes available and occupied respectively. The transition $\mathcal{X}^t_{f,o}(p)$ is naturally defined by $1 - \mathcal{X}^t_{f,a}(p)$. In our setting, we assume that the current resources' states are fully observable, i.e. we get real-time information whether a resource is available or occupied. For most applications the graph G is a street network and the expected travel costs w for traversing one edge can be estimated by: $^{distance}/_{speed\_limit} + penalty$, were $penalty$ is a hyperparameter that specifies the expected additional costs that occur, i.e. waiting time for turning, or waiting for a green traffic light. In our experiments we set this value to 30s if the agent turns and 0s in all other cases. Additionally, we consider the terminal costs c_w for occupying a resource. For the parking use-case, the terminal costs might represent the walking time from the parking spot to the goal which can be computed by the distance divided by the average walking speed. The DRRP now aims to guide a user to a free resource while minimizing the expected cost to claim an available resource and reach the destination.

4 MDP Model for DRRP

We focus on infinite horizon MDPs that minimize the expected costs (in contrast to maximizing the expected rewards). The MDP used here is a five tuple (S, A, T, C, γ). S is a finite set of states and A is a finite set of actions. For each state $s \in S$ there may be only a subset of actions $A(s) \subseteq A$ applicable. The known Markovian transition function $T^a_{s,s'} \in [1, 0]$ defines the probability that the MDP changes from state s to s' after the action a was executed. $C : S \times A \to \mathbb{R}$ is a function defining the costs for being in a state and taking an action. The discount factor γ defines the farsightedness of an agent, i.e. the reward of k steps in future will be discounted by the factor γ^k. In a general MDP, the discount factor can have the range of $0 \leq \gamma < 1$, however, in specific MDPs, like Stochastic Shortest Paths (see below), the discount factor can be 1 as well.

A policy $\pi : S \to A$ defines an action to choose for every state. Usually, the goal is to minimize the expected costs. The optimal policy π^* is the policy that defines the action $\pi^*(s) \in A(s)$ (i.e. the best action) for every state s that minimizes the expected discounted future costs U (also called utility):

$$U(s) = E\left[\sum_{k=0}^{\infty} \gamma^k \, C(s_k, \pi(s_k)) \mid s_0 = s\right]$$

Any policy π yields to the corresponding utility values U^π, which can be calculated by the Bellman equation [Bellman, 1957]:

$$U^\pi(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} T^{\pi(s)}_{s,s'} \, U^\pi(s') \qquad (1)$$

The utility values that are defined by the optimal policy π^* are called optimal utility U^*.

The dynamic resource routing problem (DRRP) is modeled as follows. A state in the MDP is a $|P| + 1$ tuple, i.e. $(e, p_1, p_2, p_3, \ldots)$ where $e \in E$ is the edge the agent currently traverses and p_i is the i^{th} resource's state (available/occupied). This results in a state space growing exponentially with $O(|E| \cdot 2^{|P|})$. Additionally, we have one more state - the terminal state s_t. The actions possible at state s, $A(s)$, are defined by the outgoing edges of the current state edge's target nodes. In other words, the agent can choose one edge of all possible next edges. If the current state's edge has an available resource assigned, there is one more action, called the *take resource action*. Moreover, if the agent chooses this action, it will traverse with probability *one* to the terminal state. This means that the agent can choose weather it wants to take a resource or not. The state transition function T can be defined for all states where $next(s.e, a) = s'.e$ and

a is not the *take resource action* by the transition function $\mathcal{X}$ as follows:

$$T^a_{s,s'} = \prod_{p \in P} \mathcal{X}^{W(s.e)}_{s.p.f,s'.p.f}(p)$$

For all other states, the transition probability is zero. Note that in this formula we make the reasonable assumption, that the resource transitions are independent. The amount of states s' where the transition probability $T^a_{s,s'}$ is not zero, is exactly $2^{|P|}$ for each non-terminal state s and non-*take resource action* a. The cost function C is basically defined by the costs for traversing the edge $W(e)$ plus the $penalty$. Whenever an agent takes a resource (by choosing the *take resource action*), the terminal costs c_w for the taken resource will be added. Since we do not want to discount future costs, we set the discount factor γ to *one* for the rest of this work.

The most common approaches to compute optimal utility values or an optimal policy are Value Iteration (VI) [Bellman, 1957] and Policy Iteration (PI) [Howard, 1960]. However, the computational complexity of these approaches increases very fast with the size of the state space. Hence, these methods are infeasible for on the fly computations in the DRRP.

[Bertsekas and Tsitsiklis, 1991] introduced the Stochastic Shortest Path (SSP) problem which is basically a MDP with additional restrictions. First, all costs need to be strictly greater than zero. Additionally, there exists a non-empty set of initial states $\mathcal{S} \subset S$ and a non-empty set of goal-states $\mathcal{G} \subset S$. Finally, there exists at least one proper policy for this MDP, that is, a policy that eventually reaches one state in $\mathcal{G}$ with probability one.

The DRRP can be seen as a subset of the stochastic shortest path problem. The underlying MDP contains one terminal state. Every state where the agent's position equals the position of a currently available resource, allows an action that stands for occupying the resource and traversing into the terminal state with probability one. As long as there is at least one resource whose state transits with a probability greater than zero to available, there exists a proper policy - visiting this resource until it is available. Thus, DRRPs can be solved by specialized algorithms for stochastic shortest paths. We base our solution on RTDP for two reasons. First, RTDP does not consider states which will not be contained in any competitive solution. Imagine an environment, where an agent starts the search at the very north of a road network and is looking for a resource at the center of the network. In this case, all states where the agent is located at the very south of the network will obviously never occur in any competitive solution. The second reason is that with RTDP we have anytime results available. Therefore, we can get good (yet not optimal) approximations during early steps of the search, while the algorithm is still improving the policy in the background.

RTDP, however, requires the MDP to hold another condition, namely that all non-proper policies incur infinite costs for at least one state. This condition is satisfied due to the fact all costs are greater than zero and there is a final set of states implying loops in the policy (since the terminal state will not be reached).

The RTDP algorithm is a trail-based dynamic programming approach. A trail starts at a random start state of $\mathcal{S}$ and greedily chooses the best action from the actual utility values. If the utility has not yet been set for this state yet, a heuristic approximation (lower bound) will be used instead. This heuristic has a dual purpose. First it should lead the greedily acting update process to the goal states without having to touch all possible states. Second, the closer the heuristic is to the real utility values U the less updates need to be executed until the utility converges to the optimum.

When an action is chosen, the utility value will be updated and the current state switches to a state which is sampled according to the probability distribution T. This procedure will be repeated until the terminal state is reached or a maximum number of steps has been performed. Once the trail is finished, another trail will be executed.

Though RTDP improves the policy fast in the first trails, it takes a long time until it converges completely. This is due to the fact that we are sampling according to the probability distribution. However, at some point the likely states have been converged already to a non-optimal value and updates of less likely states are required to fully converge to the global optimum. In theory RTDP needs to compute a infinite number of trails. The Bounded RTDP (BRTDP) algorithm [McMahan *et al.*, 2005] (see Algorithm 1) overcomes this issue by introducing an additional upper bound heuristic. Instead of sampling according to T, the BRTDP algorithm additionally considers the difference between the upper and lower bound, resulting in that the probability for states with a high gap between upper and lower bound become more probable. One additional benefit of BRTDP is that it comes with an error estimate (upper minus lower bound). Thus we can terminate the algorithm as soon as the difference of the start states' utility values are lower than our accepted error rate α. The τ hyperparameter can be set to anything greater than 1 and has little effect on the runtime [McMahan *et al.*, 2005].

Algorithm 1 Bounded Real-time Dynamic Programming

procedure BRTDP(State s_0)
  while $(U^u(s_0) - U^l(s_0)) > \alpha$ **do**
    trail(s_0)
procedure TRAIL(State s_0)
  $s \leftarrow s_0$
  while $s \notin \mathcal{G}$ **do**
    // if U is not set for a state $\Rightarrow$ use heuristic value
    $U^u(s) \leftarrow \min_{a \in A(s)} C(s,a) + \gamma \sum_{s' \in S} T^a_{s,s'} U^u(s')$
    $a \leftarrow \text{argmin}_{a \in A(s)} \sum_{s' \in S} T^a_{s,s'} U^l(s)$
    $U^l(s) \leftarrow \min_{a \in A(s)} C(s,a) + \gamma \sum_{s' \in S} T^a_{s,s'} U^l(s')$
    $\forall s' \in S : b(s') = (U^u(x) - U^l(x)) \cdot T^a_{s,s'}$
    $B \leftarrow \sum_{s' \in S} b(s')$
    if $B < U^u(x) - U^l(x))/\tau$ **then** break
    $s \leftarrow$ sample from distribution $b(s')/B$

5 Bound Heuristic Improvements

5.1 Lower bound heuristic

To lower bound the utility for a state s, we propose to compute the shortest path costs from the current location of the agent to the resource offering the best combined cost between traveling and terminal costs. Assuming that the resource is available at arrival, this obviously yields a solution with minimal costs. Let us note that this bound is similar to the lower bound poroposed in [McMahan *et al.*, 2005]. However, the general bound in [McMahan *et al.*, 2005] computes the shortest path in the state space whereas our heuristic is based on the Graph G. Thus, the size of underlying graph for computing the lower bound shrinks by the factor $2^{|P|}$.

5.2 Upper bound heuristic

[McMahan *et al.*, 2005] have introduced an upper bound that works on every SSP, called Dijkstra Sweep for Monotone Pessimistic Initialization (DS-MPI). However, this bound computes values for many states that may not be touched by the BRTDP algorithm at all. Even worse, the DS-MPI algorithm has a time complexity of $O(|S| \cdot \log(|S|))$ (assuming a constant branching factor) which is huge in our problem setting. Recall that $|S|$ grows with $O(|E| \cdot 2^{|P|})$ and the branching factor grows with $O(2^{|P|})$. In the following, we introduce an adapted upper bound for the DRRP, called Minimum Expected Wait Time (MEWT).

The MEWT upper bound computes for every resource the expected travel time that is necessary if the agent only considers this particular resource. In particular, we compute the time for traveling to the resource and add the expected waiting time until the resource will be available. However, we do not consider an action that allows the agent to wait in front of the resource. Though, this is generally possible, we decided against it because it usually is not an option which is allowed for many cases. Therefore, we compute the optimal path to return to the respective resource with a minimum travel time (minimum round trip time). The worst case complexity of this shortest path computation is $O(|N| \log(|N|) + |E|)$. Fortunately, this path is usually very short and not dependent on the size of the graph G. Thus, the computation time is negligible in the majority of cases. Given the minimum round trip time, we can compute the probability that the resource is available after each round trip employing $\mathcal{X}$. Using this probability, we derive the expected number of round trips necessary until the spot is available. Thus, the expected waiting time is computed as follows:

$$t_{\text{wait}}(p) = t_{rt}(p) \cdot 1/\mathcal{X}_{o,a}^{t_{rt}(p)}(p) \tag{2}$$

Here, $t_{rt}(p)$ denotes the minimum round trip time for resource p. The overall upper bound can be computed for all edges that are not part of any round trip by the following formula:

$$\min_{p \in \mathcal{P}} [t_{\text{wait}}(p) + \textit{dijkstra}(n, p) + c_w(p)] \tag{3}$$

For all edges that are part of any resource's round trip, the state of the relevant resources need to be considered in order to hold the monotonicity constraint of the upper bound.

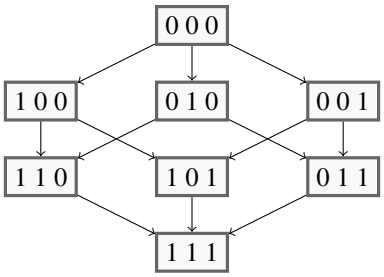

Figure 1: Search graph (directed acyclic graph) for a setting with three resources. The nodes of this graph contain the value of t_o.

The underlaying behavior is a valid, proper policy and therefore, an upper-bound to the optimal policy as well. The computing complexity of this upper bound is $O(|P| \cdot (|N| \log(|N|) + |E|))$ and $O(|N| \log(|N|) + |E|)$ in the general case for all states in S.

6 Approximations

Often multiple resources are placed on the same edge of the graph G. For example, this is very common for drop-off stations or on-street parking spots. An agent does not care if there is one or all resources available. What matters is whether there is at least one resource available to claim. In order to reduce the number of resources and thus, the state space, we track whether or not a spatially clustered set of resources has at least one available resource. Thus, we summarize all resources on an edge and compute the probability that not all resources on the edge are currently occupied. To integrate this method into our model, we modify the transition likelihoods in the Markov models describing the state transitions. Thus, all resources on the same edge are modelled by a single Markov model with states $no - resource - available$ and $any - resource - available$.

As described in section 4 the number of state transitions $|\{s' | s' \in S \wedge T_{s,s'}^a > 0\}|$ for each state $s \notin \mathcal{G}$ and non-*take resource action* a is $2^{|P|}$. However, most of those transitions are very unlikely. Imagine the case that all resources are available. The probability that in the next step all resources will be occupied is usually extremely small, even though not zero. Therefore, we propose to only consider a reduced set of transitions which include the most likely cases. In the following, we show how we can efficiently compute the most likely transitions in decreasing order such that the sum of their probabilities is greater than $1 - \epsilon$ without having to touch every possible state. In other words, we accept a maximum error of ϵ by considering the most likely transitions only while pruning a large amount of very unlikely transitions.

The basic idea of this algorithm is to separate the non-stochastic edge transition of the street graph G from the resource transitions, defined by $\mathcal{X}$. For the sake of simplicity, we ignore the certain edge transition in the following. Since we assume that the resource transitions are independent, it is possible to handle each resource's binary transition distribution separately. The most probable MDP transition is obviously the transition where the most likely event occurs for every resource transition. To compute further events

yielding the highest probabilities in a monotonically decreasing order, we introduce two bit-arrays t and t_o plus an array that contains the current probabilities of the resources' state r. Whereas t represents whether it is more likely that a resource is available or not, t_o describes the current event, i.e. whether a resource changed its state or not. The resulting algorithm is basically a best first search on a directed acyclic graph (see figure 1 for a setting with three resources). The nodes of this graph contain the values of t_o. We invert all probabilities that are lower than 0.5 by computing $r(p) = 1 - r(p)$ and mark the respective resource in t by setting the bit to *one*. Now, the product of all values in r is the probability for the most likely event. The state of the resources that produces this probability can be recalculated using t.

Now, we want to compute the next highest probabilities. It is obvious that the second highest transition probability has exactly *one* value in r inverted (computing $1 - x$). In order to retrieve this event, we make use of t_o in the following expansion step. This bit-array remembers which values in t_o have been inverted. We insert all possible bit-arrays that can be constructed by setting exactly one *zero* to *one* of the current bit-array t_o into a priority queue. The priority is the product of all values of r while taking the inversion defined by the new bit-array t'_o into account (c.f. function *prob* in algorithm 2). The top of the priority queue retrieves the event with the second highest transition probability.

This procedure can be repeated in order to retrieve always the event having the highest probability. In the following, we prove that this algorithm (c.f. Algorithm 2) computes the transition probabilities in a decreasing monotonic order.

Lemma 1. *The probabilities of the expanded events are always smaller or equal to the parent node.*

Proof. An expanded event differs from the parent event by changing one *zero* to a *one*. This means that we exchange a more likely event with a less likely event. The corresponding probability of the more likely resource state x is $0.5 \leq P(x) \leq 1$, while the probability of the less likely state x' is $1 - P(x)$ and thus $0 \leq P(x') \leq 0.5$. Due to the independence assumption the overall probability is the product of the probabilities of all assumed states. Now that we have exchanged only one state we can compute the new overall probability by multiplying the factor $P(x')/P(x)$. Since we know that $P(x') \leq P(x)$ it is implied that the expanded overall probability is less or equal to the probability of the parent event. □

Lemma 2. *All possible events will be eventually handled for* $\epsilon = 0$.

Proof. The root of the search directed acyclic graph (see figure 1) is the zero vector, which is obviously contained. On the first layer, we expand all vectors that have exactly one resource state toggled. In the expansion step, each event in the n^{th} layer expands events that contain $n + 1$ *ones*, namely the already existing *ones* plus one more *one*. Since all combinations are available in the n^{th} layer, it is obvious that all combinations will be expanded in the $n + 1^{th}$ layer. □

Algorithm 2 Input: ϵ, probability list r_0 of free resources, output: a list of resource state arrays with a probability $> 1-\epsilon$

```
procedure TopProbabilities(ε, r_0)
    t, r ← transform(r_0)
    result ← empty list
    closed Set([0, 0, ...])
    queue PriorityQueue(([0, 0, ...], prob([0, 0, ...], r)))
    while queue not empty do
        t_o, p_max ← poll(queue)
        result add (XOR(t_o, t), p_max)
        if probsum(result) > 1 − ε then
            break
        for i in range(|P|) do
            if t_o[i] = 0 then
                t'_o ← t_o.clone
                t'_o[i] ← 1
                if t'_o not in closed then
                    closed add t'_o
                    queue add (t'_o, prob(t'_o, r))
    return result
```

Lemma 3. *When all events will be handled (Lemma 2) and the expansion retrieves always smaller probabilities (Lemma 1), the priority queue retrieves always the highest probable event left, i.e. the algorithm retrieves all events in decreasing order with respect to its probability.*

Proof. Assume that there is an event with a higher probability than the event at the top of the queue, given Lemma 1 and 2. We know that this event will be handled eventually (Lemma 2). Then this event must be either in the queue, which is impossible due to the fact that it is a priority queue, or it must be in any expansion set of not yet expanded nodes. However, this implies that an expanded child must have higher probability than its parent which is in conflict with Lemma 3 (proof by contradiction). □

Due to the fact that we are modifying the probabilities during the approximation, we have to adapt the MEWT upper bound heuristic. Since $\mathcal{X}$ is modeled by a continuous time Markov chain, the probability $\mathcal{X}^{t_{rt}}_{o,a}(p)$ from formula 2 can be divided as follows:

$$\begin{aligned}\mathcal{X}^{t_{rt}}_{o,a}(p) = P_a(t_d) \cdot \mathcal{X}^{t_{rt}-t_d}_{a,a}(p) \\ + P_o(t_d) \cdot \mathcal{X}^{t_{rt}-t_d}_{o,a}(p)\end{aligned}$$

Here $P_a(t_d)$ denotes the probability that the resource is available at any given time point $0 < t_d < t_{rt}$. Analogously, P_o is the probability for being occupied at a given time point. Notwithstanding, during the approximation we modify the Markov chain at time point t_d, yet not more than ϵ. In the worst case for our upper bound we reduce P_a and P_o by the factor $(1 - \epsilon)$. The worst case estimate for $\mathcal{X}$, called $\hat{\mathcal{X}}$, can be computed by the following formula:

$$\begin{aligned}\hat{\mathcal{X}}^{t_{rt}}_{o,a}(p) &= (1-\epsilon) \cdot P_a(t_d) \cdot \mathcal{X}^{t_{rt}-t_d}_{o,a}(p) \\ &\quad + (1-\epsilon) \cdot P_o(t_d) \cdot \mathcal{X}^{t_{rt}-t_d}_{a,a}(p) \\ &= (1-\epsilon) \cdot \mathcal{X}^{t_{rt}}_{o,a}(p)\end{aligned}$$

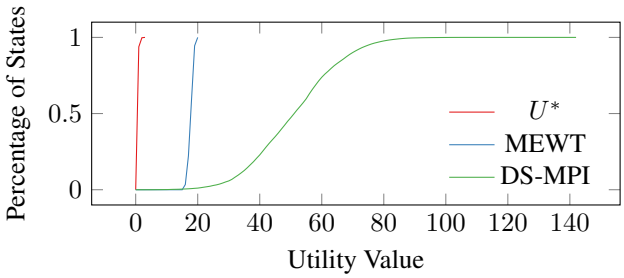

Figure 2: Comparison of the quality of the upper bounds MEWT and DS-MPI on a graph with 2148 nodes, 4057 edges and five randomly chosen resources. The y-axis shows the percentage of states that have at most the given (x-axis) utility value. By way of comparison the optimal utility value are plotted as well.

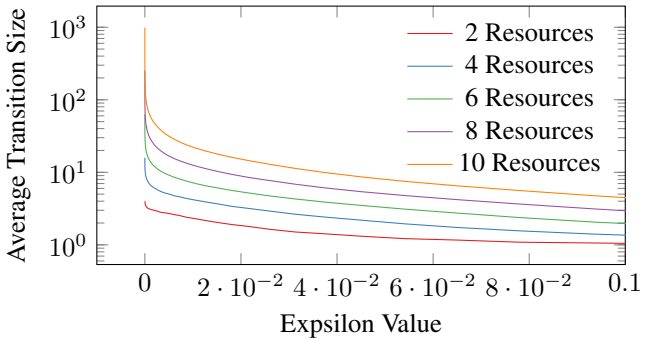

Figure 3: The amount of transitions with respect to ϵ are plotted in this figure. Note that the y-axis is log scaled. The street graph of this experiment has 204 nodes and 339 edges.

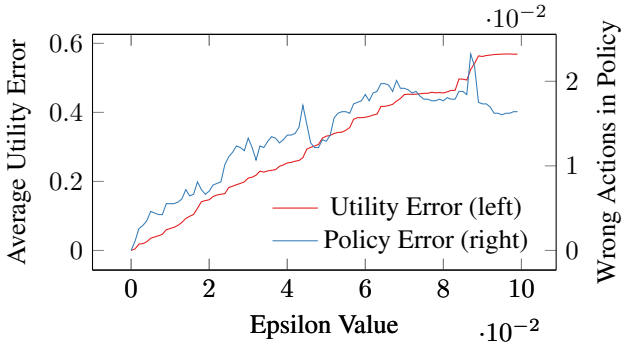

Figure 4: This experiment evaluates the utility and policy error caused by the approximation. The values for utility error are printed on the left hand side and the fraction of wrong values in the policy on the right hand side. The street graph of this experiment has 204 nodes and 339 edges.

This procedure can be repeated recursively for arbitrary many time points and we receive the following formula for n approximations:

$$t_{\text{wait}}(p) = t_{rt}(p) \cdot 1/(\mathcal{X}_{o,a}^{t_{rt}(p)}(p)(1-\epsilon)^n) \qquad (4)$$

7 Evaluation

Our evaluation was executed on the following system: Linux 4.4.0-109-generic x86_64 GNU/Linux, 8x Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz, 32GB DIMM DDR3 Synchron 1600 MHz.

In order to evaluate the quality of the MEWT upper bound, we compare it to the DS-MPI upper bound from [McMahan *et al.*, 2005]. The experimental setting uses a sub-graph of Melbourne with 2148 nodes and 4057 edges and five randomly chosen edges that contain resources. Figure 2 plots the percentage of states that have at most the respective value on the x-axis. The utility values for U^*, MEWT and DS-MPI are visualized. Obviously U^* reaches the percentage of *one* (100%) most rapidly. MEWT grows at the beginning a little slower than DS-MPI. However, at the utility value 16 the MEWT upper bound has more states that have at least this utility value than the DS-MPI upper bound. All utility values from MEWT are below 20 while DS-MPI has utility values from up to 142. Thus, we can see that for most states, the utility value of MEWT is closer to U^* than DS-MPI and hence, the quality of the bound is better for DRRP.

For the next experiment, we computed the average amount of states that are possible (probability greater than zero) to occur after being at a specific state and choosing an action. The street network is a sub-graph of Melbourne with 204 nodes and 339 edges. The assumed expected time for an available resource becoming occupied is three minutes and the assumed expected time for an occupied resource becoming available is seven minutes for all randomly chosen resources. Figure 3 plots the average transition size (y-axis) with respect to an epsilon value (x-axis). Please note that the y-axis is log scaled. As can be seen, the transition function T contains a huge percentage of follow-states that have a very low probability and thus, they have numerically no big weight in the sum of the bellman equation (Algorithm 1). This effect is even more strengthened when the amount of resources increases. Hence, small epsilons yields a performance improvement with negligible utility errors.

We evaluated the error caused by this approximation in the following experiment. We computed the optimal utility values using value iteration on the same subgraph of Melbourne and computed the average utility error as follows:

$$\frac{1}{|S|}\sum_{s \in S} |U^*(s) - U^{\pi_\epsilon^*}(s)|$$

Here, $U^{\pi_\epsilon^*}$ is the optimal utility that will be computed by the MDP that uses the approximation with the given epsilon value ϵ. The utility value *one* represents the expected travel time of *one* minute in this experiment. The policy error is computed by the fraction of the best actions of the approximated optimal policy that deviate from the actual optimal policy. Figure 4 illustrates both measures with increasing epsilon values (x-axis). The graph suggests that the errors are linearly increasing for low epsilon values up to $\epsilon = 0.1$.

The experiment depicted in figure 5 evaluates the runtime of the BRTDP algorithm with MEWT upper bound for varying the epsilon values. As can be seen, the runtime decreases significantly once the epsilon rate is bigger than zero. In this setting, a very small epsilon (0.001) reduces the runtime by a

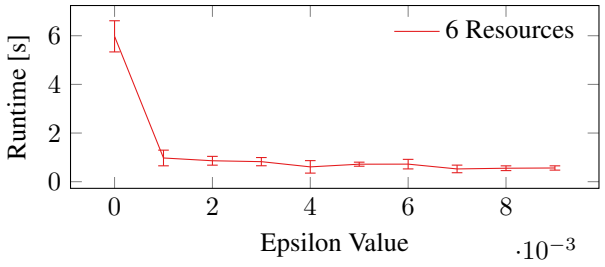

Figure 5: Runtime of the BRDP algorithm with MEWT and transition approximation for increasing ϵ values(x-axis) on a graph with 204 nodes, 339 edges and six resources.

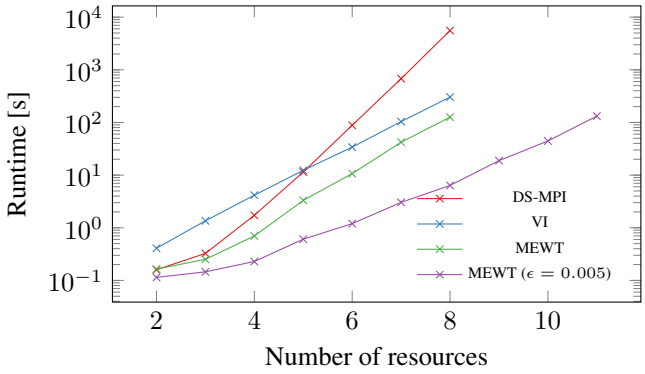

Figure 6: Average runtime of different approaches (Value Iteration, BRTDP with DS-MPI/MEWT/MEWT with $\epsilon = 0.005$) over 20 runs on a graph having 204 nodes and 339 edges. The resources and the start state were randomly chosen for each run.

factor of six. At the same time figure 4 shows that an epsilon as small as 0.001 does not significantly change the utility values and thus the policy.

Figure 6 compares the runtime of the different approaches. This experiment was executed on a sub-graph of Melbourne with 204 nodes and 339 edges. Please note that the y-axis (runtime) of those figures are log scaled. The figure illustrates that the MEWT upper bound with epsilon value 0.005 outperforms the other approaches by a order of magnitude for larger numbers of resources.

8 Conclusion

In this paper, we contributed adapted bounds for BRTDP for the dynamic resource routing problem with fully observable resources. The BRTDP algorithm performs significantly faster with the introduced MEWT upper bound due to its lower computational complexity and its better quality (as empirically shown). Moreover, we have introduced a approximation technique which we empirically show to reduce the computation time of the optimal policy by another order of magnitude. In future research we want to examine whether partially observable MDPs can be applied in a setting where the resources' states are not fully observable. Furthermore, we want to develop different model-free reinforcement learning approaches for this problem setting.

References

[Barto *et al.*, 1995] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[Bellman, 1957] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[Bertsekas and Tsitsiklis, 1991] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[Bonet and Geffner, 2003a] Blai Bonet and Hector Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, pages 1233–1238, 2003.

[Bonet and Geffner, 2003b] Blai Bonet and Hector Geffner. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *ICAPS*, volume 3, pages 12–21, 2003.

[Guo and Wolfson, 2016] Qing Guo and Ouri Wolfson. Probabilistic spatio-temporal resource search. *GeoInformatica*, pages 1–29, 2016.

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

[Howard, 1960] Ronald A Howard. Dynamic programming and markov processes.. 1960.

[Jossé *et al.*, 2015] G. Jossé, K. A. Schmid, and M. Schubert. Probabilistic resource route queries with reappearance. In *#EDBT15#*, 2015.

[McMahan *et al.*, 2005] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd international conference on Machine learning*, pages 569–576. ACM, 2005.

[Puterman, 1994] Martin L Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

[Sanner *et al.*, 2009] Scott Sanner, Robby Goetschalckx, Kurt Driessens, Guy Shani, et al. Bayesian real-time dynamic programming. In *IJCAI*, pages 1784–1789. Citeseer, 2009.

[Schmoll and Schubert, 2018] Sebastian Schmoll and Matthias Schubert. Dynamic resource routing using real-time information. In *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.*, pages 501–504, 2018.

[Smith and Simmons, 2006] Trey Smith and Reid Simmons. Focused real-time dynamic programming for mdps: Squeezing more out of a heuristic. In *AAAI*, pages 1227–1232, 2006.