Resource-based Dynamic Rewards for Factored MDPs

Ronan Killough, Kim Bauters, Kevin McAreavey, Weiru Liu, Jun Hong
School of EEECS
Queen's University Belfast
Belfast, UK
Email: {rkillough01, k.bauters, kevin.mcareavey, w.liu, j.hong}@qub.ac.uk

***Abstract*—Factored MDPs provide an efficient way to reduce the complexity of large, real-world domains by exploiting structure within the state space. This avoids the need for the state space to be fully enumerated, which is impractical in large domains. However, defining a reward function for state transitions is difficult in a factored MDP since transitions are not known prior to execution. In this paper, we provide a novel method for deriving rewards from information within the states in order to determine intermediate rewards for state transitions. We do this by treating some specific state variables as *resources*, allowing costs and rewards to be inferred from changes to the resources and ensuring the agent is resource-aware while also being goal-oriented. To facilitate this, we propose a novel variant of Dynamic Bayesian Networks specifically for modelling action transitions and capable of dealing with relative changes to real-valued state variables (such as resources) in a compact fashion. We also propose a number of reward functions which model resource types commonly found in real-world situations. We go on to show that our proposed framework offers an improvement over existing techniques involving reward functions for factored MDPs as it improves both the efficiency and decision quality of online planners when operating on these models.**

I. INTRODUCTION

Markov Decision Processes (MDPs) [2] have become a standard framework for modelling stochastic planning problems [11]. In this paper, we focus on factored state MDPs, where each state is not atomic, but is an instantiation of a set of random variables S [3]. As with all MDPs, we must define a transition function to describe the stochastic movement between states upon taking an action. We should also define a reward function to guide the agent's behaviour. Defining these functions is straightforward when we have a fully defined state space as it allows an agent designer to describe both transition probabilities and transition rewards explicitly. However, when the state space becomes very large or even infinite, it becomes impractical to describe these functions explicitly for each transition. While factored MDPs are a very convenient tool for describing such state spaces, the transition and reward functions are more difficult to define compactly, while still being expressive.

A common approach to defining a reward function for a factored MDP is to have a set of "local" reward functions, each dependent on a small subset of state variables. For each state transition, these rewards are calculated given the outcome state and combined additively to produce a reward for that transition [10], [14]. An issue with this approach is that it requires an agent designer to assign "value" to complex changes in state variables. This can often be difficult to quantify and can lead to arbitrary value assignments and thus sub-optimal solutions. In this paper, we overcome this issue by introducing a novel class of MDP which relies on "resources" as the units of value within the model. Resources are, at their most general, continuous random variables held as part of the state S. By rewarding acquisition and penalising consumption of resources, we can easily define reward models with very little input from an agent designer.

The value of desirable states is not always directly related to a resource gain or loss, usually because they have value which is external to the domain. We can refer to such states as *goals*. One simple way to represent the value of goals is in terms of the number of resources we are willing to expend in order to achieve them. When goal values are directly related to resource levels, the agent can easily determine a cost-benefit relationship for the goal, and decide to what degree that goal is worth pursing. It also allows for applications where multiple goal states are present since the value of all goals are known in relation to the resources they consume. This, however, is left for future work.

While resources are a good indicator of the "value" of a state transition, in real-world scenarios, resources are often both *constrained* and *finitely useful*. Generally, they are constrained because they can run out, and finitely useful because only so much of them are needed to achieve the goal. We explore cases with both of these properties by having the reward value for a resource *change* be dynamically dependent on the resource *level* in the prior state.

The transition function of a factored MDP is usually represented as a Dynamic Bayesian Network (DBN) [3], [16]. For each action, probabilistic dependencies between variables in the outcome state and the prior state are defined. Having resources as part of the state involves dealing with changes to continuous variables. Most works dealing with factored MDPs only consider absolute changes to binary or enumerated variables [3], [15]. In order to have resources influence the reward function, our DBNs must be able to represent *relative* changes to continuous variables [4]. We therefore introduce a novel form of DBNs referred to as Action-DBNs which can compactly represent conditional, stochastic effects to continuous state variables between two states.

Throughout the paper we refer to a running example of a planetary rover. The rover must operate under both time and power constraints, *i.e.* its resources. The rover begins in an initial location and has three actions. It can (1) Wait and gain power at the cost of time; (2) Sleep overnight and gain time at the cost of power and a mission day; and (3) Move to a "sampling" location. By moving it may also become stuck. Once at the sampling location the rover has a choice of sampling procedures, Sample1 and Sample2, which cost uncertain amounts of time and power. The resources here are constrained, an action cannot be performed unless there are sufficient resources to do so. The goal of the agent is to complete both sampling procedures (defined as the goal state) without running out of time, power, or mission days.

The paper is organised as follows. In Section II we introduce MDPs and factored MDPs, as well as approximate online planning methods for solving MDPs. We show the ability of the framework to compactly deal with real-valued state variables in the transition function in Section III. In Section IV we define a number of approaches for dynamically producing reward values based on resource consumption and availability. In Section V we evaluate our framework on various domain configurations. Finally, in Sections VI and VII we discuss related work and provide concluding remarks.

II. PRELIMINARIES

In this section we provide some required preliminaries on MDPs and specifically factored MDPs. We also discuss Dynamic Bayesian Networks (DBNs), specifically in their application as a model of the transition function for a factored MDP. The approximate planning methods; Monte Carlo Tree Search (MCTS) and UCT [9], are also introduced. These are state of the methods for online planning and are used to evaluate the proposed methods.

A. MARKOV DECISION PROCESSES

A finite-state MDP consists of a set of states S, a set of actions A, a transition function T and reward function R. The transition function is defined as $T : S \times A \times S \to [0, 1]$, *i.e.* given a state s and an action a, we transition to a new state s' with probability $T(s, a, s')$. Rewards are typically associated with state transitions and are defined as $R : S \times A \times S \to \mathbb{R}$, with $R(s, a, s')$ the reward for taking action a in state s and arriving in state s'. To determine the best action to take, we must consider the immediately available actions and their outcomes, as well as all future possible actions and outcomes up to some horizon h. This horizon may be fixed to a certain number of steps. Equivalently, we may consider a discount factor δ which prioritises the influence of immediate outcomes over those in the future while also ensuring the sum is finite [8].

Generally, we want to produce a mapping between states and actions which will tell the agent how to behave in any given state. This mapping is referred to as a *policy*. Since the objective is to maximise the overall expected reward, an optimal policy will consistently select actions which will maximise both immediate and potential future rewards. We can define a policy as $\pi_t : S \to A$ with t as the current time step. The quantitative *value* of a policy π with an initial state s_0 is given by the function $V(\pi(s_0))$:

$$V(\pi(s_0)) = E\left(\sum_{t=0}^{h-1} \delta^{\,t} \cdot R(s_t, \pi_t(s_t), s_{t+1}) \right)$$

Here, $\pi_t(s_t)$ produces an action according to policy π_t. The optimal policy π^* will maximise $V(\pi(s_0))$, which is the expected reward of taking actions according to π beginning at state s_0 up to horizon h.

Factored MDPs allow the representation of large state spaces by having states comprised of a set of state variables $S = \{S_0 \dots S_n\}$ where each variable $S_i \in S$ can take a value in its domain Δ_i. We define a state ω as a function $\omega : S \to \Delta$ where $\omega(S_i) = s_i$ is an instantiation of a variable S_i in the state ω and $\Delta = \Delta_0 \cup \dots \cup \Delta_n$. The number of possible states in a factored MDP is therefore $\prod_{S_i \in S} |\Delta_i|$. Since the states are not atomic, it becomes possible to exploit structure in the domain to more compactly represent both the transition and reward functions. Specifically, the transition function can be modelled using a Dynamic Bayesian Network.

A Bayesian network is a directed acyclic graph which describes dependencies (graph edges) between variables (graph nodes) [6]. When the graph edges correspond to changes to the values of variables over *time*, we refer to it as a dynamic-BN or DBN. We can model a single state transition (from state ω to state ω') as the two "layers" of this graph, where each $s'_i \in \omega'$ (represented as a node) has a set of dependencies on variables in the previous state ω or on other variables within ω'. These sets are referred to as the *parents* of each s'_i or $Pa(s'_i)$. For each node with dependencies, a conditional probability distribution (CPD) must be defined to describe the stochastic relationship between that node and its parents. The probability of the transition from ω to ω' given an action can then be calculated as $\prod_{s' \in \omega'} P\big(s' | Pa(s')\big)$. We denote a two-layer DBN as $\{\omega_0, \dots, \omega_n, \omega_0^{'}, \dots, \omega_n^{'}\}$. This specific type of DBN is often referred to as a two time-slice BN or a 2TBN.

B. APPROXIMATE PLANNING METHODS

Even when the state space of an MDP is finite, establishing optimal policies for MDPs is often infeasible for all but the smallest domains [7]. Therefore, in recent years, attention has shifted to approximate methods such as Monte Carlo Tree Search (MCTS). Planners based on MCTS can be referred to as *online* or *anytime* planners, since they can return an approximation of the next best action when required, with accuracy improving over time. MCTS is a best-first search algorithm which uses the reward values in the MDP to build up more promising parts of the tree, and ignore branches which are expected to offer low reward.

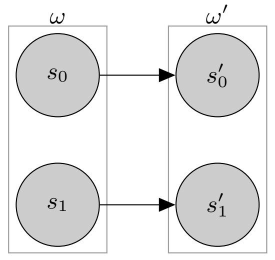
The process of MCTS can be described as 4 distinct steps. The algorithm first *selects a leaf node* based on some selection function, the search is then *expanded* by adding a new random child node, a *random playout* of the domain is then carried out to some terminal state, and finally the rewards associated with each node are *propagated back up the tree* to the root node. The UCT algorithm extends this approach by incorporating a selection mechanism where new nodes are selected based on both their expected reward and the number of times they have been visited. This allows the search to cover more promising parts of the tree as well as parts which have rarely been visited. For this reason, UCT is well suited for very large search spaces, such as those generally associated with factored MDPs [5].

III. DEFINING STATE TRANSITIONS WITH CONTINUOUS VARIABLES

In this section we introduce a compact method of dealing with relative changes to continuous variables between states. This is necessary to define transition functions for MDPs with state spaces containing resources, since we are often trying to describe a static increase or decrease to these variables [4]. As mentioned in Section II-A, a specific variant of DBN, the 2TBN, is used in much of the literature to represent state transitions in factored MDPs. Many such works only consider state spaces with binary variables [3], [15]. In these cases, standard 2TBNs are adequate to represent the transitions as there can only be two conditions and two effects per variable, *i.e.* a CPD for a transition of $s \rightarrow s'$ will only have $2^{|Pa(s')|}$ sets of probabilities. When dealing with continuous variables, such as those used to represent resources, it becomes impractical to define CPDs which exhaustively describe the probabilities of each outcome instance of s'. For this reason, we introduce a novel variant of 2TBNs which allow relative changes to continuous variables to be represented compactly. Since these graphs are designed specifically to describe the effects and conditions of an action, we refer to this as action-DBNs or ADBNs.

Example 1: Given a state $S = \{S_0, S_1\}$ where $\Delta_0 = \{True, False\}$ and $\Delta_1 = \mathbb{Z}$, we first define a transition function of a given action as a 2TBN [13]. Consider a single action which has the effect of setting S_0' to $True$ provided s_0 is $False$, this happens with a probability of 0.8. The action also sets S_1' to $s_1 + 2$ provided that s_1 is currently less than 10. Otherwise, S_1' will be set to 0. Figure 1 shows a transition graph with the directed edges representing dependencies between variables. State ω' represents state ω in the successive time step, after the execution of an action. As can be seen from Figure 1, defining a transition function for the Boolean variable S_0 is trivial, however defining the transition condition on the continuous variable S_1 is impossible to do exhaustively, even for this small example.

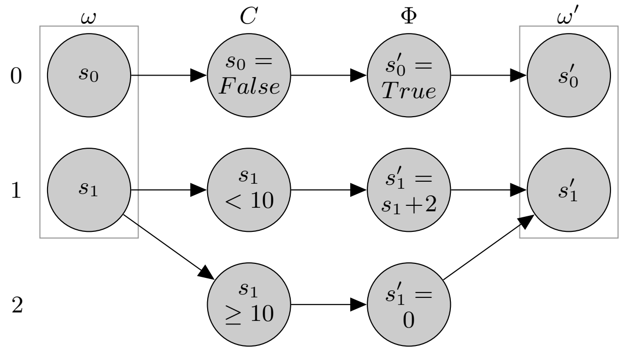
To allow for a compact representation of transitions such as $s_1 \rightarrow s_1'$ shown in Example 1, we introduce two additional layers to the 2TBN. The first is a set of conditions, C. These are propositions on state variables. The second layer is a set of effects, Φ, which can be applied to variables in ω to produce ω'. By reducing these relative continuous relationships to a set of conditions and effects, the CPDs need only define Boolean relationships, as the *effect* is either applied to the variable in the previous state, or it is not.



s_0 \ s_0'	True	False
True	1	0
False	0.8	0.2

s_1 \ s_1'	...	9	10	11	12	...
⋮		⋮	⋮	⋮	⋮	
8	...	0	1	0	0	...
9	...	0	0	1	0	...
10	...	0	1	0	0	...
⋮		⋮	⋮	⋮	⋮	

Fig. 1: A 2TBN and associated CPDs



c_0 \ ϕ_0	True	False
True	0.8	0.2
False	1	0

c_1 \ ϕ_1	True	False
True	1	0
False	0	1

c_2 \ ϕ_2	True	False
True	1	0
False	0	1

Fig. 2: An Action-DBN (The numbering at the left indicates each node's index within the sets.) and associated CPDs

The 2TBN described in Example 1 and its associated CPDs are shown in Figure 2 as an ADBN. For s_1, we simply state that effect ϕ_1 is applied (True) with a probability of 1, if and only if condition c_1 holds. Each effect $\phi \in \Phi$ can have any number of dependencies upon conditions $c \in C$. We refer to this set of conditions as the *parents* of the effect, denoted $Pa(\phi)$.

It is also possible for effects to have dependencies on other effects, such relationships are referred to as *synchronic arcs*. This allows variables in ω' to be dependent on the outcome of other variables in that state. For example, the effect ϕ_0 from the example above could instead have parents $Pa(\phi_0) =$

$\{s_1 < 10, \phi_1 = True\}$. This would mean that s'_1 will only be increased by 2 in the (stochastic) eventuality that s'_0 ends up as $False$.

In order to calculate the probability of a set of effects being applied to a state ω given an ADBN, we first determine the set of effects which apply given the previous state ω and the set of conditions, referred to as Φ^a where $\Phi^a \subseteq \Phi$. The effect transition probability is as follows:

$$T(\omega, a, \Phi^a) = \prod_{\phi^a \in \Phi^a} P\big(\phi^a | Pa(\phi^a), \omega\big)$$

where $P\big(\phi^a | Pa(\phi^a)\big)$ is the probability of ϕ^a being applied. Applying this set of effects to the prior state ω will produce the outcome state ω'.

IV. RESOURCE-BASED REWARD FUNCTIONS

Many approaches for solving factored MDPs rely on decomposed reward functions. In these cases, a set of localised reward functions R_j is defined, with each function dependent on a subset of individual state variables $S^{R_j} \subseteq S$. This allows the reward associated with reaching a state ω to be calculated as $R(\omega) = \sum_{j=1}^{n} R_j(S_i^{R_j})$ [3], [10]. While this is useful for incentivising the agent's goal, it is generally not practical for defining intermediate rewards in order to provide guidance towards more promising paths, especially when the number of possible transitions is extremely large or even infinite. This problem is further compounded by how difficult it is to define reward relationships between variables over state transitions in such a way that they are isomorphic with goal-based rewards.

We begin by defining a certain subset of the state S as *resources*. This set of resource variables is denoted as V where $V \subseteq S$. A resource is any continuous state variable to which increases and decreases can be considered *gains* and *costs*, respectively. In many cases, it will be useful to have resources which are constrained (can run out), however this is not an essential property by our definition. If we consider our planetary rover example, its "power" is a resource. It is always desirable to increase the power level of the agent. However, since this is a constrained resource; even if the power level at the goal state is greater than zero, dropping to zero at any step means that the agent cannot proceed towards its goal. For this reason, having intermediate rewards to keep track of the resource levels at each step is vital.

A. DEFINING GOALS

To rely entirely on resources to determine our reward would limit the behaviour of the agent purely towards minimising average resource loss. Since the least expensive path in terms of resources will not typically lead towards the defined goal, the agent will therefore not be goal-oriented. We consider domains in which the agent must reach some goal state whilst also minimising resource consumption. Therefore, a goal state shall influence a separate reward function $R^g : \Omega \to \mathbb{R}$. We propose a novel method of defining goal rewards which are isomorphic with the rewards related to resource consumption. While this method would also apply to domains with multiple goals, we only consider domains with a single goal in this work.

A goal g is a tuple $\langle \Gamma, REV \rangle$. Γ is a condition over some subset of S. Since each state is an instantiation of the set of variables, it is desirable to define the goal as a condition over the variables instead of as an explicit state. If a state ω' satisfies this condition then that state is a goal state. We define a language $\mathcal{L}$ to express a condition Γ as follows.

- If $s \in S_i, \delta \in \Delta_i$ and $\circ$ is some binary relation ($=, \neq, <, >, \leq$ or $\geq$) over Δ_i, then $s \circ \delta$ is an (atomic) formula.
- If Γ is a formula, then $\neg\Gamma$ is a formula.
- If Γ and Γ' are formulas, then $\Gamma \wedge \Gamma$ and $\Gamma \vee \Gamma$ are formulas.

The state function is extended as $\omega : \mathcal{L} \to \{true, false\}$. A state $\omega \in \Omega$ is a goal state with respect to its goal condition $\Gamma \in \mathcal{L}$, denoted $\Gamma \models \omega$, iff $\omega(\Gamma) = true$ in the standard truth-functional way. The set of all states which satisfy the goal condition is $\{\omega \in \Omega | \Gamma \models \omega\}$.

When a goal state is reached, the agent receives an additional reward beyond what is associated with the resource change for that state transition. This reward is what provides incentive for the goal. To ensure goal and resource-based rewards are isomorphic in their values, we define goal rewards in terms of the number of resources we are willing to expend to achieve them. This is the REV component of our goal definition. REV is a set of *resource equivalent values*, where each $q_V \in REV$ is a *resource amount*. This amount corresponds to how much of the associated resource the agent is willing to expend (from the initial state) in order to achieve the goal. This is not an indicator of how much resources an agent designer *expects* the goal to consume, but acts as an upper, worst-case limit. In this way, *we define the value of a goal in terms of resources*. The goal reward associated with arriving in state ω for a *single* resource V_i is defined as:

$$R^g_{V_i}(\omega) = \begin{cases} q_{V_i} \in REV, & \text{if } \Gamma \models \omega \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Example 2: Consider the planetary rover example. The state consists of seven variables: *location, time, power, days, stuck, sampled1* and *sampled2*. We have a single goal definition g. The condition Γ associated with this goal g is Γ = { *time*≥0 ∧ *power*≥0 ∧ *days*≥0 ∧ *sampled1*=True ∧ *sampled2*=True}. Given these conditions, it is clear that a potentially infinite number of states can be deemed goal states, provided they satisfy all conditions in Γ.

The definition of goal state rewards in Equation 1 produces some interesting behaviour, in that as soon as the cumulative resource cost of reaching a goal has exceeded its "worth" (as determined by its REVs), then the agent will no longer pursue this goal. This behaviour is especially useful in domains pervaded by uncertainty, as well as domains with multiple goals, since the agent can determine that the *expected* resource cost of achieving the goal exceeds its assigned worth, and could therefore be said to either be too risky, or indeed be of lower priority than another goal.

B. STATIC REWARD FUNCTIONS

In order to have our agent guided by resource gain and loss, we must consider particular properties of resources when designing our reward function. The simplest case is resources which are not constrained. That is, resources cannot run out and have no upper limit. In this case the relationship between reward and resource change is linear, as it is always equally desirable to gain resources and equally undesirable to lose them, regardless of the current resource level. Since we consider a set of resources, each resource may not have equal value per unit, meaning each resource must be weighted according to its relative importance. One way to do this is to provide a reference point V^{ref} indicating a desirable or comfortable level of this resource. Dividing the resource change by this reference point will appropriately weight the resources according to relative importance. The following formula is a reward function for unconstrained resources.

$$R(\omega, a, \omega') = \sum_{V_i \in V} v_\omega - (v_{\omega'} + R^g_{V_i}(\omega)) \frac{1}{V_i^{ref}} \frac{1}{|V|} \tag{2}$$

where v_ω and $v_{\omega'}$ are the respective levels of resource V in state ω and ω'. Using this reward function, we can consider our agent to be simultaneously pursuing two objectives. Note that the goal reward $R^g_{V_i}(\omega)$ is added to the resource level at state ω'. Therefore, if ω' is a goal state, the primary objective is to reach the goal state(s) and the secondary objective is to consume as few resources as possible while doing so. Having the *resource equivalent values* (REVs) as part of the goal definition ensures an appropriate balance is struck between the two objectives since they provide a comparable measure of value between the goal and the resources.

C. DYNAMIC REWARD FUNCTIONS

While the reward function in the previous section will accurately represent the reward associated with resource gains and losses, it will not be accurate when resources are constrained. In constrained environments, resources can run out or max out. Often, resources are only required in finite amounts, meaning they become less valuable if they are obtained in excess of what is needed. We describe three types of constraints which can apply to a resource.

- **Resource exhaustion**. This means the resource can run out. When it is close to doing so, the reward function should penalize losses more heavily than when it is abundant.
- **Resource limitation**. This means the agent has an upper limit on the amount of a resource it can obtain, *i.e.* a finite capacity. In this case, we simply stop rewarding gains if they exceed the agent's capacity for this resource.
- **Resource saturation**. This means that the agent only has a finite requirement for a resource, and therefore should receive diminishing returns on resource gains beyond this requirement.

These resource properties can act as heuristics that allow the planner to avoid branches which acquire unneeded amounts of resources as well as branches whose resource levels run dangerously low.

Given the resource information from the prior and outcome states of a transition (V_ω and $V_{\omega'}$) we also define a static reference point V^{ref} for each resource which describes a comfortable or desirable level of this particular resource. Our reward function is therefore defined as $R_v \rightarrow S_i \times S_i \times \mathbb{R}$. It is V^{ref} which allows the agent to determine which levels of a resource V are considered either *scarce* or *abundant*.

We first set out a number of desirable principles that a resource-based dynamic reward function should adhere to. Such principles are only relevant to constrained resources.

Principle 1: $V^i_\omega = V^i_{\omega''} \implies R_v(v^i_\omega, v^i_{\omega'}, V^{ref}_i) + v^i_{\omega'}, v^i_{\omega''}, V^{ref}_i) = 0$

Principle 1 requires that given two states with the same resource availability, the aggregate reward received for a resource from all intervening transitions should sum to zero. This removes any incentive for the agent to throw away resources and subsequently gain them back to maximise reward.

Principle 2: $(V^{ref}_i = V^{ref}_j) \wedge (v^i_{\omega'} - v^i_\omega) = (v^j_{\omega'} - v^j_\omega) \wedge v^i_\omega > v^j_\omega \implies |R_v(v^i_\omega, v^i_{\omega'}, V^{ref}_i)| < |R_v(v^j_\omega, v^j_{\omega'}, V^{ref}_j)|$

Principle 2 ensures that given two resources, the resource with the lower prior availability should receive a greater *absolute* reward, assuming the resource change and reference point are the same for both resources.

Principle 3: $V^{ref}_a < maxV^{ref} < V^{ref}_b \implies R_V(v_\omega, v_{\omega'}, V^{ref}_a) < R_V(v_\omega, v_{\omega'}, maxV^{ref}) > R_V(v_\omega, v_{\omega'}, V^{ref}_b)$

Principle 3 governs the setting of V^{ref}, and states that given a resource change and availability, there shall be a set reference point $maxV^{ref}$ at which this change provides the greatest reward to this availability level. For a V^{ref} less than $maxV^{ref}$, the availability is considered excessive and thus the reward will be reduced. For values greater than the maximum the change is considered less significant relative to what is required to achieve the goal.

We now introduce three reward functions which model resources with various combinations of constraints. We must determine a "sufficiency" level K_V between 0 and 1 for each resource at a given state ω. This is necessary since reward will be dependent on the availability of the resource when that resource is constrained. When the resource availability is equal to the reference point V^{ref}, the sufficiency will equal 0.5. A higher sufficiency will indicate excessive availability and a lower sufficiency level will indicate a limited supply. We calculate the sufficiency level K for a resource V in state ω as:

$$K_V(v_\omega, V^{ref}) = -\Big(\frac{1}{v_\omega}\Big)^{\frac{1}{\frac{log(V^{ref})}{log(2)}}} + 1 \tag{3}$$

The reward for each resource V_i is calculated as the difference between the sufficiency level of the resource in state ω and ω'. How the sufficiency level is used is dependent on the properties of the resource. We introduce three reward functions for resources with various combinations of properties. Rewards from these resource-specific functions can then be combined.

$$R_V(v_\omega, v_{\omega'}, V^{ref}) = \begin{cases} K_V(v_\omega, V^{ref}) - K_V(v_{\omega'}, V^{ref}), & \text{if } v_{\omega'} - v_\omega < 0 \\ (v_{\omega'} - v_\omega)\frac{1}{V_i^{ref}}, & \text{otherwise.} \end{cases} \quad (4)$$

$$R_V(v_\omega, v_{\omega'}, V^{ref}) = \begin{cases} K_V(v_\omega, V^{ref}) - K_V(v_{\omega'}, V^{ref}), & \text{if } v_{\omega'} - v_\omega < 0 \\ \lambda - v_\omega, & \text{if } v_{\omega'} > \lambda \\ (v_{\omega'} - v_\omega)\frac{1}{V_i^{ref}}, & \text{otherwise.} \end{cases} \quad (5)$$

$$R_V(v_\omega, v_{\omega'}, V^{ref}) = K_V(v_\omega, V^{ref}) - K_V(v_{\omega'}, V^{ref}) \quad (6)$$

We also include a number of caveats to the reward functions above which are as follows.

1) If the action depletes the resource to 0 or below ($v_{\omega'} \leq 0$), reward is always -1.
2) If the action effect sums to 0 ($v_{\omega'} - v_\omega = 0$), reward is always 0.
3) If the resource availability is 0 or less ($v_\omega \leq 0$) and the action effect is positive ($v_{\omega'} - v_\omega > 0$), reward is always 1.

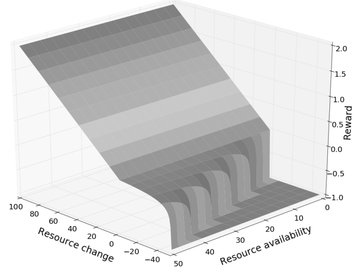


Fig. 3: A reward function for an exhaustible resource.

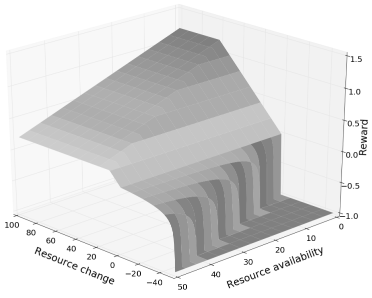


Fig. 4: A reward function for an exhaustible resource with an upper limit.

Figures 3, 4 and 5 plot the three reward functions defined in equations 4, 5 and 6 respectively. They each plot reward for a single resource V given resource changes ranging from -50 to +100 and resource availability ranging from 0 to 50. In all cases, V^{ref} is equal to 50. Figure 3 shows a reward function for a resource which is exhaustible, while Figure 4 shows one which is both exhaustible and has an upper limit λ, which in this case is 60. Figure 5 shows a resource which is both exhaustible and can be saturated. Changes which deplete this resource to very low levels are heavily penalised and gains to the resource are less valuable if the current availability is higher.

To calculate the reward $R(\omega, a, \omega')$ associated with the overall state transition, we additively combine the resource-based and goal-based reward functions for each resource and then take the average of the rewards across all resources:

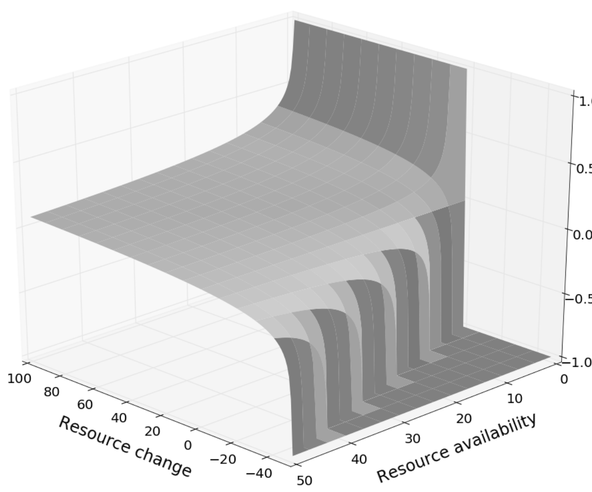


Fig. 5: A reward function for an resource which can be both exhausted and saturated.

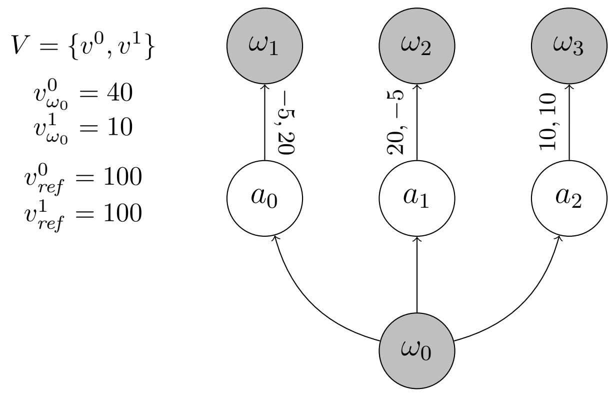
$$R(\omega, a, \omega') = \Big(\sum_{V_i \in V} R_{V_i}\big(v_\omega, v_{\omega'}, V^{ref}\big) + R_{V_i}\big(v_{\omega_0}, v_{\omega_0} + R^g_{V_i}(\omega'), V^{ref}\big)\Big)\frac{1}{|V|} \quad (7)$$

The reward function is therefore dependent on both the availability in the preceding state ω, the effect of the action $(v_{s'} - v_s)$ and whether or not the outcome state is a goal. Note that the goal reward $R^g_{V_i}(\omega')$ is not affected by the *current* availability of the resource, but by its availability in the initial state v_{ω_0}. This is because the *value* of the goal is defined as an amount of the resource to be expended from the initial state. As can be seen in Figures 3, 4 and 5; the bounds on reward values are dependent on the resource properties. Generally $R(\omega, a, \omega') \in [-1, 1]$, however if the resource cannot be saturated, rewards can exceed 1 for gains which exceed the current availability level.

These functions reflect the intuitive notion, as formalised in Principle 2, that gains/losses should be rewarded/punished more severely when resources are scarce and also that gains to abundant resources should reap diminishing returns.

Example 3: Figure 6 shows a simple example of how optimal decisions can be determined when the reward function is based on constrained resources. The agent begins in state ω_0 and has a choice of three actions which will allow it to transition to one of three outcome states $\omega_1, \omega_2, \omega_3$. Each action a_0, a_1, a_2 has a different effect on the two resources. These resources (v^0 and v^1) are both constrained. Assuming two separate agents p_0 and p_1. The agents are identical except for their reward function. Agent p_0 uses a static reward function, whereas agent p_1 uses our proposed dynamic reward function as described above. For agent p_1 therefore, the availability of the resources in state ω will influence the reward obtained from state transitions.

The table in Figure 6 shows the rewards obtained for each action by each of the two agents. Agent p_0 simply favours the



	a_0	a_1	a_2
p_0	7.5	7.5	10
p_1	0.096	-0.044	0.089

Fig. 6: A simple decision example.

highest aggregate resource gain (action a_2), since it is blind to resource availability. Agent p_1 recognises that these resources are constrained, and therefore actions which maximise gain to the scarcer resource (v^1 in this case) are prioritised. The best action for p_1 is therefore a_0, which offers a large gain to the scarcer resource, despite coming at a larger or equal cost to v^0 compared to alternative actions. Action a_1 receives an overall negative reward, since the cost to the scarcer resource is not outweighed by the larger gain to the more abundant one.

V. Evaluation

We will now show how a dynamically influenced reward function offers an improvement to both the efficiency and decision quality of an online planner by more accurately modelling the relationship between the domain's resources and the reward obtained. We compare four different reward functions, including our proposed function. The rover domain is used throughout. The four reward functions we compare are as follows:

1) **Dynamic resource-based reward**. This is our proposed function where the reward is based on the resources consumed, the resource availability, and the proximity to goal states.
2) **Static resource-based reward**. In this case the reward is linearly related to the resource change, it is not affected by the resource availability. Goal rewards are simply treated as resource gains and combined with any resource related reward.
3) **Goal-only reward** In this case the agent is simply awarded a +1 for reaching a goal state. No intermediate rewards are present. This approach is commonly used in domains where the aim is to reach the goal in the fewest possible steps.
4) **State-based reward** Here, a static reward is placed on goal states and states which indicate movement toward the goal. Static costs are placed on "failure" states where the agent can no longer continue.

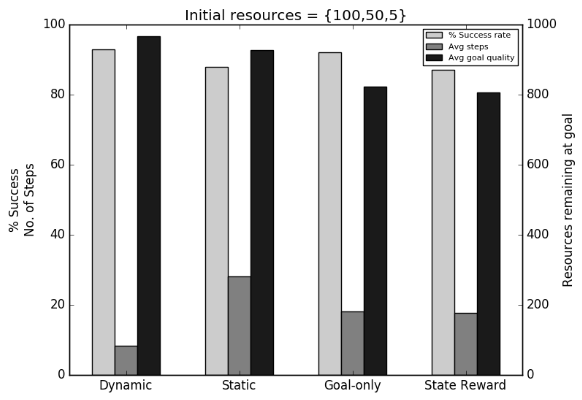


Fig. 7: Comparing reward functions.

In all cases we use a UCT-based online planner adapted to operate on factored MDPs by generating new states on-the-fly by applying actions. These actions are defined as ADBNs, encoding probabilistic dependencies between state variables, as outlined in Section III. The rover domain contains discrete uncertainty in action outcomes, both in resource costs from sampling actions and the possibility of becoming stuck when moving. For all experiments we run a set number of full-domain playouts involving multiple planning phases from an initial state to the goal state or failure. We analyse three criteria to assess plan quality:

- **Average success rate** The number of times the agent successfully reached a goal state. The agent can fail either by becoming stuck or by running out of any of the three resources.
- **Average steps to goal** The average number of actions taken to reach the goal. Due to uncertainty in the domain, the minimum number of steps required to reach the goal can vary, however this can show if the agent took a large number of unnecessary steps.
- **Average goal quality** Goal quality is defined as the *weighted* aggregate of remaining resources at the goal state.

When calculating goal quality, each time unit is worth 2 power units and each day unit is worth 100 time units. The reason for these weights is due to the resource-exchanging actions. The *wait* action provides 20 power units at a cost of 10 time units and the *sleep* action provides 100 time units at the cost of a single day unit. For both average steps taken and average goal quality, our results show the average of successful attempts only.

We run 100 full-domain playouts from an initial state with 100, 50 and 5 in *resp.* time, power, and day units. Each planning phase has a 0.5s run. This can be considered a high initial resource level as it is sufficient to reach the goal in all but the worst cases. The results are shown in Figure 7. Here, the proposed planner (labelled "Dynamic") offers both the highest success rate and the best goal quality on average in terms of resources consumed, it can therefore be said to be both goal-oriented and resource-efficient. The static resource-based planner offers only slightly worse levels of resource efficiency, however by prioritising resource acquisition it suffers a higher failure rate.

VI. Related Work

In [3], an approach is proposed which allows the construction of optimal policies for large MDPs, without full enumeration of the state space by using structured policies applied to states which satisfy a set of conditions. It makes use of 2-TBNs to represent dependencies between states. Non-optimal solutions to factored first-order MDPs are explored in [15]. Here, factored transitions and additive rewards are specified in a way which can easily scale with domain size and allow for approximate solutions to be found. Both these papers attempt to find solutions to MDPs with extremely large or infinite state spaces. However they only deal with scenarios where state variables are binary. It is both possible and practical to represent *absolute* stochastic transitions with these frameworks. As with much of the literature concerning factored MDPs, the ability to represent relative changes to variables is not explicitly considered.

A more complex reward function is presented in [1]. This paper deals specifically with a electricity smart-grid domain, where "prosumers" can both buy and sell electricity from the grid. The aim is to minimize financial cost of the power used while minimizing stress on storage systems (batteries). The reward function captures this by rewarding low tariff purchases while punishing excessive charging/discharging of the batteries. Despite an incredibly large state space, this domain could be solved optimally using value iteration, largely due to the bounding and discretisation of state variables. However, run time for a solution was multiple hours. This is acceptable in this domain since optimisation is carried out for each 24-hour period.

Planning with continuous resources is addressed in [12] by using a hybridised version of the AO* algorithm. This allows both continuous and discrete state variables to be dealt with. This work limits the state space by applying heuristics which avoid planning trajectories through which goal states are not accessible. Such heuristics are especially useful here since resource-constraints can significantly limit reachability. This approach is shown to be capable of dealing with enormous state spaces which include continuous components, however it does place some constraints not present in our framework, namely that resources are non-replenishable and non-exchangeable. This means that excessive resource acquisition is not an issue for the proposed algorithm.

VII. Conclusions

In this paper, we proposed a compact method for dealing with relative state transitions to continuous variables in factored MDPs. We did this by proposing a new variant of DBNs aimed specifically at modelling state transitions. We also explored methods for using resources as the basis for reward functions in factored MDPs. By adapting reward functions to fit various typical properties of resources, we show that they can influence agent behaviour in reliable and intuitive ways, meanwhile limiting the burden on the agent designer of guiding behaviour toward desirable states. We proposed a novel method for assigning goal rewards in terms of "resource worth", which enables an agent to effectively balance resource aware and goal driven behaviours.

By applying our methods to an example domain, we show that we can improve resource efficiency and success rate by accurately modelling the relationship between resource consumption and reward via the proposed reward function.

For future work we would like to explore the capability of this framework to handle multiple goals. Since goals are defined in terms of resource consumption, the "value" of multiple goals can be compared and re-evaluated in terms of resources expended so-far. Furthermore, risk awareness would be a valuable addition to the proposed framework as exhaustible resources can have catastrophic consequences for the agent if they run out, such risks should be considered in decision making.

References

[1] A. Angelidakis and G. Chalkiadakis. Factored MDPs for optimal prosumer decision-making. In *Proc. of AAMAS'15*, pages 503–511, 2015.

[2] R. Bellman. A Markovian decision process. Technical report, DTIC Document, 1957.

[3] C. Boutilier, R. Dearden, M. Goldszmidt, et al. Exploiting structure in policy construction. In *IJCAI*, volume 14, pages 1104–1113, 1995.

[4] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, et al. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 77–84. Morgan Kaufmann Publishers Inc., 2002.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on comp. intell. and AI in games*, 4(1):1–43, 2012.

[6] T. Dean and K. Kanazawa. *A model for reasoning about persistence and causation*. Brown University, Department of Computer Science, 1989.

[7] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, pages 399–468, 2003.

[8] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *AI*, 101(1):99–134, 1998.

[9] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proc. of ECML'06*, pages 282–293, 2006.

[10] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *IJCAI*, volume 99, pages 1332–1339, 1999.

[11] D. Koller and R. Parr. Policy iteration for factored MDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 326–334. Morgan Kaufmann, 2000.

[12] N. Meuleau, E. Benazera, R. I. Brafman, E. A. Hansen, and M. Mausam. A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*, 34(1):27, 2009.

[13] K. P. Murphy. *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

[14] P. Poupart, C. Boutilier, R. Patrascu, and D. Schuurmans. Piecewise linear value function approximation for factored MDPs. In *AAAI/IAAI*, pages 292–299, 2002.

[15] S. Sanner and C. Boutilier. Approximate solution techniques for factored first-order MDPs. In *ICAPS*, pages 288–295, 2007.

[16] A. L. Strehl, C. Diuk, and M. L. Littman. Efficient structure learning in factored-state MDPs. In *AAAI*, volume 7, pages 645–650, 2007.