# Equi-Reward Utility Maximizing Design in Stochastic Environments

**Sarah Keren**[†] **Luis Pineda**[‡] **Avigdor Gal**[†] **Erez Karpas**[†] **Shlomo Zilberstein**[‡]

[†]Technion–Israel Institute of Technology

[‡]College of Information and Computer Sciences, University of Massachusetts Amherst

**Abstract**

We present the Equi-Reward Utility Maximizing Design (ER-UMD) problem for redesigning stochastic environments to maximize agent performance. ER-UMD fits well contemporary applications that require offline design of environments where robots and humans act and cooperate. To find an optimal modification sequence we present two novel solution techniques: a compilation that embeds design into a planning problem, allowing use of off-the-shelf solvers to find a solution, and a heuristic search in the modifications space, for which we present an admissible heuristic. Evaluation shows the feasibility of the approach using standard benchmarks from the probabilistic planning competition and a benchmark we created for a vacuum cleaning robot setting.

## Introduction

We are surrounded by physical and virtual environments with a controllable design. Hospitals are designed to minimize the daily distance covered by staff, computer networks are structured to maximize message throughput, human-robot assembly lines are designed to maximize productivity, *etc.* Common to all these environments is that they are designed with the intention of maximizing some user benefit while accounting for different forms of uncertainty.

Typically, design is performed manually, often leading to far from optimal solutions. We therefore suggest to automate the design process and formulate the *Equi-Reward Utility Maximizing Design* (ER-UMD) problem where a system controls the environment by applying a sequence of modifications in order to maximize agent utility.

We assume a fully observable stochastic setting and use Markov decision processes (Bellman 1957) to model the agent environment. We exploit the alignment of system and agent utility to show a compilation of the design problem into a planning problem and piggyback on the search for an optimal policy to find an optimal sequence of modifications. In addition, we exploit the structure of the offline design process and offer a heuristic search in the modifications space to yield optimal design strategies. We formulate the conditions for heuristic admissibility and propose an admissible heuristic based on environment simplification. Finally, for settings where practicality is prioritized over optimality, we present a way to efficiently acquire sub-optimal solutions.

The contributions of this work are threefold. First, we formulate the ER-UMD problem as a special case of *environment design* (Zhang, Chen, and Parkes 2009). ER-UMD supports arbitrary modification methods. Particularly, for stochastic settings, we propose modifying probability distributions, an approach which offers a wide range of subtle modifications. Second, we present two new approaches for solving ER-UMD problems, specify the conditions for acquiring an optimal solution and present an admissible heuristic to support the solution. Finally, we evaluate our approaches given a design budget, using probabilistic benchmarks from the International Planning Competitions, where a variety of stochastic shortest path MDPs are introduced (Bertsekas 1995) and on a domain we created for a vacuum cleaning robot. We show how redesign substantially improves utility, expressed via reduced cost achieved with a small modification budget. Moreover, the techniques we develop outperform the exhaustive approach reducing calculation effort by up to $30\%$ .

The remaining of the paper is organized as follows. Section  describes the ER-UMD framework. In Section , we describe our novel techniques for solving the ER-UMD problem. Section  describes an empirical evaluation followed by related work (Section ) and concluding remarks (Section ).

## Equi-Reward Utility Maximizing Design

The *equi-reward utility maximizing design* (ER-UMD) problem takes as input an environment with stochastic action outcomes, a set of allowed modifications, and a set of constraints and finds an optimal sequence of modifications (atomic changes such as additions and deletions of environment elements) to apply to the environment for maximizing agent expected utility under the constraints. We refer to sequences rather then sets to support settings where different application orders impact the model differently. Such a setting may involve, for example, modifications that add preconditions necessary for the application of other modifications (e.g. a docking station can only be added after adding a power outlet).

We consider stochastic environments defined by the quadruple $\epsilon = \langle S_\epsilon, A_\epsilon, f_\epsilon, s_{0,\epsilon} \rangle$ with a set of states $S_\epsilon$, a set of actions $A_\epsilon$, a stochastic transition function $f_\epsilon : S_\epsilon \times A_\epsilon \times S_\epsilon \rightarrow [0,1]$ specifying the probability $f(s,a,s')$ of reaching state $s'$ after applying action $a$ in $s \in S$, and an initial state $s_{0,\epsilon} \in S_\epsilon$. We let $\mathcal{E}$, $\mathcal{S}_\mathcal{E}$ and $\mathcal{A}_\mathcal{E}$ denote the set of all environments, states and actions, respectively.

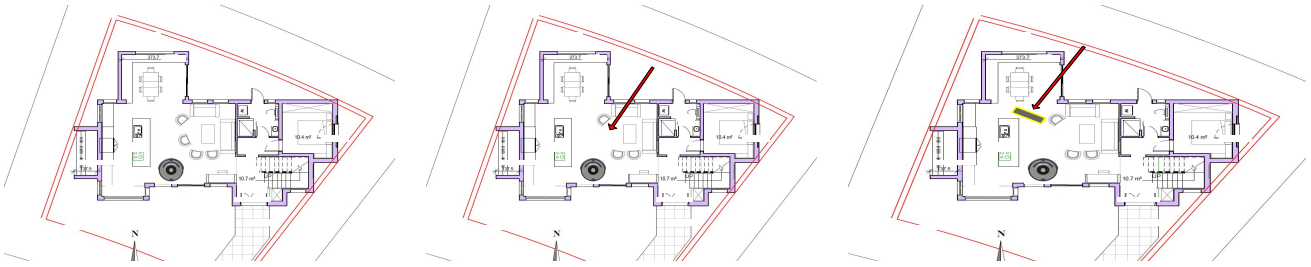

Figure 1: An example of an ER-UMD problem

Adopting the notation of Zhang and Parkes (2008) for environment design, we define the ER-UMD model as follows.

**Definition 1** *An equi-reward utility maximizing (ER-UMD) model $\omega$ is a tuple $\langle \epsilon^0_\omega, \mathcal{R}_\omega, \gamma_\omega, \mathbf{\Delta}_\omega, \mathcal{F}_\omega, \Phi_\omega \rangle$ where*

- $\epsilon^0_\omega \in \mathcal{E}$ *is an initial environment.*
- $\mathcal{R}_\omega : \mathcal{S}_\mathcal{E} \times \mathcal{A}_\mathcal{E} \times \mathcal{S}_\mathcal{E} \to \mathbb{R}$ *is a Markovian and stationary reward function, specifying the reward $r(s, a, s')$ an agent gains from transitioning from state $s$ to $s'$ by the execution of $a$.*
- $\gamma_\omega$ *is a discount factor in $(0, 1]$, representing the deprecation of agent rewards over time.*
- $\mathbf{\Delta}_\omega$ *is a finite set of atomic modifications a system can apply. A modification sequence is an ordered set of modifications $\vec{\Delta} = \langle \Delta_1, \ldots, \Delta_n \rangle$ s.t. $\Delta_i \in \mathbf{\Delta}_\omega$. We denote by $\vec{\mathbf{\Delta}}_\omega$ the set of all such sequences.*
- $\mathcal{F}_\omega : \mathbf{\Delta}_\omega \times \mathcal{E} \to \mathcal{E}$ *is a deterministic modification transition function, specifying the result of applying a modification to an environment.*
- $\Phi_\omega : \vec{\mathbf{\Delta}}_\omega \times \mathcal{E} \to \{0, 1\}$ *is an indicator that specifies the allowed modification sequences in an environment.*

Whenever $\omega$ is clear from the context we use $\epsilon^0$, $\mathcal{R}$, $\gamma$, $\mathbf{\Delta}$,$\mathcal{F}$, and $\Phi$. Note that a reward becomes a cost when negative.

The reward function $\mathcal{R}$ and discount factor $\gamma$ form, together with an environment $\epsilon \in \mathcal{E}$ an infinite horizon discounted reward Markov decision process (MDP) (Bertsekas 1995) $\langle S, A, f, s_0, \mathcal{R}, \gamma \rangle$. The solution of an MDP is a control policy $\pi : S \to A$ describing the appropriate action to perform at each state. We let $\Pi_\epsilon$ represent the set of all possible policies in $\epsilon$. Optimal policies $\Pi^*_\epsilon \subseteq \Pi_\epsilon$ yield an expected accumulated reward for every state $s \in S$ (Bellman 1957). We assume agents are optimal and let $\mathcal{V}^*(\omega)$ represent the discounted expected agent reward of following an optimal policy from the initial state $s_0$ in a model $\omega$.

Modifications $\Delta \in \mathbf{\Delta}$ can be defined arbitrarily, supporting all the changes applicable to a deterministic environment (Herzig et al. 2014). For example, we can allow adding a transition between previously disconnected states. Particular to a stochastic environment is the option of modifying the transition function by increasing and decreasing the probability of specific outcomes. Each modification may be associated with a system cost $\mathcal{C} : \mathbf{\Delta} \to \mathbb{R}^+$ and a sequence cost $\mathcal{C}(\vec{\Delta}) = \sum_{\Delta_i \in \vec{\Delta}} \mathcal{C}(\Delta_i)$. Given a sequence $\vec{\Delta}$ such that $\Phi(\vec{\Delta}, \epsilon) = 1$ (*i.e.*, $\vec{\Delta}$ can be applied to $\epsilon \in \mathcal{E}$ ) we let $\epsilon_{\vec{\Delta}}$ represent the environment that is the result of applying $\vec{\Delta}$ to $\epsilon$ and $\omega_{\vec{\Delta}}$ is the same model with $\epsilon_{\vec{\Delta}}$ as its initial environment.

The solution to an ER-UMD problem is a modification sequence $\vec{\Delta} \in \vec{\mathbf{\Delta}}^*$ to apply to $\epsilon^0_\omega$ that maximizes agent utility $\mathcal{V}^*(\omega_{\vec{\Delta}})$ under the constraints, formulated as follows.

**Problem 1** *Given a model $\omega = \langle \epsilon^0_\omega, \mathcal{R}_\omega, \gamma_\omega, \mathbf{\Delta}_\omega, \mathcal{F}_\omega, \Phi_\omega \rangle$, the ER-UMD problem finds a modification sequence $\vec{\Delta} \in \vec{\mathbf{\Delta}}$*

$$\underset{\vec{\Delta} \in \vec{\mathbf{\Delta}} | \Phi(\vec{\Delta}) = 1}{\textit{argmax}} \quad \mathcal{V}^*(\omega_{\vec{\Delta}})$$

We let $\vec{\mathbf{\Delta}}^*_\omega$ represent the set of solutions to Problem 1 and $\mathcal{V}^{max}(\omega) = \max_{\vec{\Delta} \in \vec{\mathbf{\Delta}} | \Phi(\vec{\Delta}) = 1} \mathcal{V}^*(\omega_{\vec{\Delta}})$ represent the maximal agent utility achievable via design in $\omega$. In particular, we seek solutions $\vec{\Delta}^* \in \vec{\mathbf{\Delta}}^*_\omega$ that minimize design cost $C(\vec{\Delta}^*)$.

**Example 1** *As an example of a controllable environment where humans and robots co-exist consider Figure 1(left), where a vacuum cleaning robot is placed in a living room. The set $\mathcal{E}$ of possible environments specifies possible room configurations. The robot's utility, expressed via the reward $\mathcal{R}$ and discount factor $\gamma$, may be defined in various ways; it may try to clean an entire room as quickly as possible or cover as much space as possible before its battery runs out. (Re)moving a piece of furniture from or within the room (Figure 1(center)) may impact the robot's utility. For example, removing a chair from the room may create a shortcut to a specific location but may also create access to a corner the robot may get stuck in. Accounting for uncertainty, there may be locations in which the robot tends to slip, ending up in a different location than intended. Increasing friction,* e.g., *by introducing a high friction tile (Figure 1(right)), may reduce the probability of undesired outcomes. All types of modifications, expressed by $\mathbf{\Delta}$ and $\mathcal{F}$, are applied offline (since such robots typically perform their task unsupervised) and should be applied economically in order to maintain usability of the environment. These type of constraints are reflected by $\Phi$ that can restrict the design process by a predefined budget or by disallowing specific room configurations.*

## Finding $\vec{\Delta}^*$

A baseline method for finding an optimal modification sequence involves applying an exhaustive best first search (BFS) in the space of allowed sequences and selecting one that maximizes system utility. This approach was used for

finding the optimal set of modifications in a goal recognition design setting (Keren, Gal, and Karpas 2014; Wayllace et al. 2016). The state space pruning applied there assumes that disallowing actions is the only allowed modification, making it non-applicable for ER-UMD, which supports arbitrary modification methods. We therefore present next two novel techniques to find the optimal design strategy for ER-UMD.

## ER-UMD compilation to planning

As a first approach, we embed design into a planning problem description. The *DesignComp* compilation (Definition 2) extends the agent's underlying MDP by adding preprocess operators that modify the environment off-line. After initialization, the agent acts in the new optimized environment.

The compilation uses the PPDDL notation (Younes and Littman 2004) which uses a factored MDP representation. Accordingly, an environment $\epsilon \in \mathcal{E}$ is represented as a tuple $\langle X_\epsilon, s_{0,\epsilon}, A_\epsilon \rangle$ with states specified as a combination of state variables $X_\epsilon$ and a transition function embedded in the description of actions. Action $a \in A_\epsilon$ is represented by $\langle prec, \langle p_1, add_1, del_1 \rangle, \ldots, \langle p_m, add_m, del_m \rangle \rangle$ where $prec$ is the set of literals that need to be true as a precondition for applying $a$. The probabilistic effects $\langle p_1, add_1, del_1 \rangle, \ldots, \langle p_m, add_m, del_m \rangle$ are represented by $p_i$, the probability of the $i$-th effect. When outcome $i$ occurs, $add_i$ and $del_i$ are literals, added and removed from the state description, respectively (Mausam 2012).

The policy of the compiled planning problem has two stages: *design* - in which the system is modified and *execution* - describing the policy agents follow to maximize utility. Accordingly, the compiled domain has two action types: $A_{des}$, corresponding to modifications applied by the design system and $A_{exe}$, executed by the agent. To separate between the stages we use a fluent *execution*, initially false to allow the application of $A_{des}$, and a no-cost action $a_{start}$ that sets *execution* to true rending $A_{exe}$ applicable.

The compilation process supports two modifications types. Modifications $\mathbf{\Delta}_X$ change the initial state by modifying the value of state variables $X_{\mathbf{\Delta}} \subseteq X$. Modifications $\mathbf{\Delta}_A$ change the action set by enabling actions $A_{\mathbf{\Delta}} \subseteq A$. Accordingly, the definition includes a set of design action $A_{des} = A_{des\text{-}s_0} \cup A_{des\text{-}A}$, where $A_{des\text{-}s_0}$ are actions that change the initial value of variables and $A_{des\text{-}A}$ includes actions $A_{\mathbf{\Delta}}$ that are originally disabled but can be enabled in the modified environment. In particular, we include in $A_{\mathbf{\Delta}}$ actions that share the same structure as actions in the original environment except for a modified probability distribution.

The following definition of *DesignComp* supports a design budget $B$ implemented using a timer mechanism as in (Keren, Gal, and Karpas 2015). The timer advances with up to $B$ design actions that can be applied before performing $a_{start}$. This constraint is represented by $\Phi^B$ that returns 0 for any modification sequence that exceeds the budget.

**Definition 2** *For an ER-UMD problem* $\omega = \langle \epsilon^0_\omega, \mathcal{R}_\omega, \gamma_\omega, \mathbf{\Delta}_\omega, \mathcal{F}_\omega, \Phi^B_\omega \rangle$ *where* $\mathbf{\Delta}_\omega = \mathbf{\Delta}_X \cup \mathbf{\Delta}_A$ *we create a planning problem* $P' = \langle X', s'_0, A', \mathcal{R}', \gamma' \rangle$ *where:*

- $X' = \{X_{\epsilon^0_\omega}\} \cup \{execution\} \cup \{time_t \mid t \in 0, \ldots, B\} \cup \{enabled_a \mid a \in A_{\mathbf{\Delta}}\}$
- $s'_0 = \{s_{0,\epsilon^0_\omega}\} \cup \{time_0\}$
- $A' = A_{exe} \cup A_{des\text{-}s_0} \cup A_{des\text{-}A} \cup a_{start}$ *where*
  - $A_{exe} = A_{\epsilon^0} \cup A_{\mathbf{\Delta}}$ *s.t.*
    - $\{\langle prec(a) \cup execution, eff(a) \rangle \mid a \in A_{\epsilon^0}\}$
    - $\{\langle prec(a) \cup execution \cup enabled_a, eff(a) \rangle \mid a \in A_{\mathbf{\Delta}}\}$
  - $A_{des\text{-}s_0} = \{\langle \langle \neg execution, time_i \rangle, \langle 1, \langle \mathrm{x}, time_{i+1} \rangle, \langle time_i \rangle \rangle \rangle \mid \mathrm{x} \in X_{\mathbf{\Delta}}\}$
  - $A_{des\text{-}A} = \{\langle \langle \neg execution, time_i \rangle, \langle 1, \langle enabled_a, time_{i+1} \rangle, time_i \rangle \rangle \mid a \in A_{\mathbf{\Delta}}\}\}$
  - $a_{start} = \langle \emptyset, \langle 1, \neg execution, \emptyset \rangle \rangle$
- $\mathcal{R}' = \begin{cases} \mathcal{R}(a), & \text{if } a \in A_{exe} \\ 0, & \text{if } a \in A_{des}, a_{init} \end{cases}$
- $\gamma' = \gamma$

Optimally solving the compiled problem $P'$ yields an optimal policy $\pi^*_{P'}$ with two components, separated by the execution of $a_{start}$. The initialization component consists of a possibly empty sequence of deterministic design actions denoted by $\vec{\Delta}_{P'}$, while the execution component represents the optimal policy in the modified environment.

The next two propositions establish the correctness of the compilation. Proofs are omitted due to space constraints. We first argue that $V^*(P')$, the expected reward from the initial state in the compiled planning problem, is equal to the expected reward in the optimal modified environment.

**Lemma 1** *Given an ER-UMD problem* $\omega$ *and an optimal modification sequence* $\vec{\Delta} \in \vec{\mathbf{\Delta}}^*_\omega$

$$V^*(P') = V^*(\omega_{\vec{\Delta}}).$$

An immediate corollary is that the compilation outcome is indeed an optimal sequence of modifications.

**Corollary 1** *Given an ER-UMD problem* $\omega$ *and the compiled model* $P'$, $\vec{\Delta}_{P'} \in \vec{\mathbf{\Delta}}^*_\omega$

The reward function $\mathcal{R}'$ assigns zero cost to all design actions $A_{des}$. To ensure the compilation not only respects the budget $B$, but also minimizes design cost, we can assign a small cost (negative reward) $c_d$ to design actions $A_{des}$. If $c_d$ is too high, it might lead the solver to omit design actions that improve utility by less than $c_d$. However, the loss of utility will be at most $c_d B$. Thus, by bounding the minimum improvement in utility from a modification, we can still ensure optimality.

## Design as informed search

The key benefit of compiling ER-UMD to planning is the ability to use any off-the-shelf solver to find a design strategy. However, this approach does not fully exploit the special characteristics of the off-line design setting we address. We therefore observe that embedding design into the definition of a planning problem results in an MDP with a special structure, depicted in Figure 2. The search of an optimal redesign policy is illustrated as a tree comprising of two component. The *design* component, at the top of the figure, describes the deterministic offline design process with nodes

representing the different possibilities of modifying the environment. The *execution* component, at the bottom of the figure, represents the stochastic modified environments in which agents act.

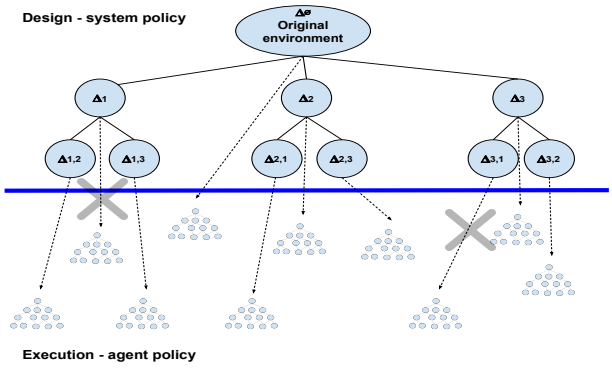


Figure 2: State space of a ER-UMD problem

Each design node represents a different ER-UMD model, characterized by the sequence $\vec{\Delta}$ of modifications that has been applied to the environment and a constraints set $\Phi$, specifying the allowed modifications in the subtree rooted at a node. With the original ER-UMD problem $\omega$ at the root, each successor design node represents a sub-problem $\omega_{\vec{\Delta}}$ of the ancestor ER-UMD problem, accounting for all modification sequences that have $\vec{\Delta}$ as their prefix. The set of constraints of the successors is updated with relation to the parent node. For example, when a design budget is specified, it is reduced when moving down the tree from a node to its successor.

When a design node is associated with a valid modification, it is connected to a leaf node representing a ER-UMD model with the environment $\epsilon_{\vec{\Delta}}$ that results from applying the modification. To illustrate, invalid modification sequences are crossed out in Figure 2.

**Algorithm 1** Best First Design (BFD)

BFD($\omega, h$)

1: create OPEN list for unexpanded nodes
2: $n_{cur} = \langle design, \vec{\Delta}_{\emptyset} \rangle$ (initial model)
3: **while** $n_{cur}$ **do**
4: **if** $IsExecution(n_{cur})$ **then**
5: **return** $n_{cur}.\vec{\Delta}$ (best modification found - exit)
6: **end if**
7: **for each** $n_{suc} \in GetSuccessors(n_{cur}, \omega)$ **do**
8: put $\langle \langle design, n_{suc}.\vec{\Delta} \rangle, h(n_{suc}) \rangle$ in OPEN
9: **end for**
10: **if** $\Phi_{\sigma}(n_{cur}.\vec{\Delta}) = 1$ **then**
11: put $\langle \langle execution, \vec{\Delta}_{new} \rangle, \mathcal{V}^*(\omega_{\vec{\Delta}_{new}}) \rangle$ in OPEN
12: **end if**
13: $n_{cur} = ExtractMax$(OPEN)
14: **end while**
15: **return** error

Using this search tree we propose an informed search in the space of allowed modifications, using heuristic estimations to guide the search more effectively by focusing attention on more promising redesign options. The *Best First Design* (BFD) algorithm (detailed in Algorithm 1) accepts as input an ER-UMD model $\omega$, and a heuristic function $h$. The algorithm starts by creating an OPEN priority queue (line 1) holding the front of unexpanded nodes. In line 2, $n_{cur}$ is assigned the original model, which is represented by a flag $design$ and the empty modification sequence $\vec{\Delta}_{\emptyset}$.

The iterative exploration of the currently most promising node in the OPEN queue is given in lines 3-14. If the current node represents an execution model (indicated by the *execution* flag) the search ends successfully in line 5, returning the modification sequence associated with the node. Otherwise, the successor design nodes of the current node are generated by $GetSuccessors$ in line 7. Each successor sub-problem $n_{suc}$ is placed in the OPEN list with its associated heuristic value $h(n_{suc})$ (line 8), to be discussed in detail next. In addition, if the modification sequence $n_{cur}.\vec{\Delta}$ associated with the current node is valid according to $\Phi$, an execution node is generated and assigned a value that corresponds to the actual value $\mathcal{V}^*(\omega_{\vec{\Delta}_{new}})$ in the resulting environment (lines 10-12). The next node to explore is extracted from OPEN in line 13.

Both termination and completeness of the algorithm depend on the implementation of $GetSuccessors$, which controls the graph search strategy by generating the sub-problem design nodes related to the current node. For example, when a modification budget is specified, $GetSuccessors$ generates a sub-problem for every modification that is appended to the sequence $\vec{\Delta}$ of the parent node, discarding sequences that violate the budget and updating it for the valid successors.

For optimality, we require the heuristic function $h$ to be admissible. An admissible estimation of a design node $n$ is one that never underestimates $\mathcal{V}^{\max}_{\omega}$, the maximal system's utility in the ER-UMD problem $\omega$ represented by $n_{cur}$.[1]

Running BFD with an admissible heuristic is guaranteed to yield an optimal modification sequence.

**Theorem 1** *Given an ER-UMD model $\omega$ and an admissible heuristic $h$, BFD$(\omega, h)$ returns $\vec{\Delta}^*_{\omega} \in \vec{\mathbf{\Delta}}^*_{\omega}$.*

The proof of Theorem 1 bares similarity to the proof of $A^*$ (Nllsson 1980) and is omitted here for the sake of brevity.

**The *simplified-environment* heuristic** To produce efficient over-estimations of the maximal system utility $\mathcal{V}^{max}(\omega)$, we suggest a heuristic that requires a single preprocessing simplification of the original environment used to produce estimates for the design nodes of the search.

**Definition 3** *Given an ER-UMD model $\omega$, a function $f : \mathcal{E} \rightarrow \mathcal{E}$ is an* environment simplification *in $\omega$ if $\forall \epsilon, \epsilon' \in \mathcal{E}_{\omega}$ s.t. $\epsilon' = f(\epsilon)$, $\mathcal{V}^*(\omega) \leq \mathcal{V}^*(\omega^f)$, where $\omega^f$ is the ER-UMD model with $f(\epsilon)$ as its initial environment.*

The *simplified-environment* heuristic, denoted by $h^{sim}$ estimates the value of applying a modification sequence $\vec{\Delta}$ to

[1] When utility is cost, it needs not to overestimate the real cost.

$\omega$ by the value of applying it to $\omega^f$.

$$h^{sim}(\omega_{\vec{\Delta}}) \stackrel{def}{=} \mathcal{V}^{max}(\omega^f_{\vec{\Delta}}) \quad (1)$$

The search applies modifications on the simplified model and uses its optimal solution as an estimate of the value of applying the modifications in the original setting. In particular, the simplified model can be solved using the *DesignComp* compilation presented in the previous section.

The literature is rich with simplification approaches, including adding macro actions that do more with the same cost, removing some action preconditions, eliminating negative effects of actions (delete relaxation) or eliminating undesired outcomes (Holte et al. 1995). Particular to stochastic settings is the commonly used *all outcome determinization* (Yoon, Fern, and Givan 2007), which creates a deterministic action for each probabilistic outcome of every action.

**Lemma 2** *Given an ER-UMD model $\omega$, applying the* simplified-environment *heuristic with $f$ implemented as an all outcome determinization function is admissible.*

The proof of Lemma 2, omitted for brevity, uses the observation that $f$ only adds solutions with higher reward (lower cost) to a given problem (either before or after redesign). A similar reasoning can be applied to the commonly used delete relaxation or any other approaches discussed above.

Note that admissibility of a heuristic function depends on specific characteristics of the ER-UMD setting. In particular, the *simplified-environment* heuristic is not guaranteed to produce admissible estimates for policy teaching (Zhang and Parkes 2008) or goal recognition design (Keren, Gal, and Karpas 2014; Wayllace et al. 2016), where design is performed to incentivize agents to follow specific policies. This is because the relaxation itself may change the set of optimal agent policies and therefore under estimate the value of a modification.

## Empirical Evaluation

We evaluated the ability to maximize agent utility given a design budget in various ER-UMD problems, as well as the performance of both optimal and approximate techniques.

We used five PPDDL domains from the probabilistic tracks of the sixth and eighth International Planning Competition[2] (IPPC06 and IPPC08) representing stochastic shortest path MDPs with uniform action cost: Box World (IPPC08/ BOX), Blocks World (IPPC08/ BLOCK), Exploding Blocks World (IPPC08/ EX-BLOCK), Triangle Tire (IPPC08/ TIRE) and Elevators (IPPC06/ ELEVATOR). In addition, we implemented the vacuum cleaning robot setting from Example 1 (VACUUM) as an adaptation of the Taxi domain (Dietterich 2000). The robot moves in a grid and collects pieces of dirt. It cannot move through furniture, represented by occupied cells, and may fail to move, remaining in its current position.

In all domains, agent utility is expressed as expected cost and constraints as a design budget. For each IPPC domain we examined at least two possible modifications, including at least one that modifies the probability distribution. Modifications by domain are specified in Table 1 with modifications marked by *change init* modify the initial state and *probability change* marks modifications to the probability function.

[2]http://icaps-conference.org/index.php/main/competitions

| | **change init** | **probability change** |
|---|---|---|
| **BOX** | relocate a truck | reduce probability of driving to a wrong destination |
| **BLOCK** | — | reduce probability of dropping a block or tower |
| **EX-BLOCK** | — | as for Blocks World |
| **TIRE** | add a spare tire at a location | reduce probability of having a flat tire |
| **ELEVATOR** | add elevator shaft | reduce probability of falling to the initial state |
| **VACUUM** | (re)move furniture | add high friction tile |

Table 1: Allowed modifications for each domain

| | B=1 | | B=2 | | B=3 | |
|---|---|---|---|---|---|---|
| | solved | reduc | solved | reduc | solved | reduc |
| BOX | 8 | 28 | 8 | 42 | 7 | 44 |
| BLOCK | 6 | 21 | 3 | 24 | 3 | 24 |
| EX-BLOCK | 10 | 42 | 9 | 42 | 9 | 42 |
| TIRE | 9 | 44 | 8 | 51 | 6 | 54 |
| ELEVATOR | 9 | 22 | 7 | 24 | 1 | 18 |
| VACUUM | 8 | 15 | 6 | 17 | 0 | — |

Table 2: Utility improvement for optimal solvers

### Optimal solutions

**Setup** For each domain, we created 10 smaller instances optimally solvable within a time bound of five minutes. Each instance was optimally solved using:

- EX- Exhaustive exploration of possible modifications.
- DC- Solution of the *DesignComp* compilation.
- BFD- Algorithm 1 with *simplified-environment* heuristic using delete relaxation and *DesignComp* to simplify and optimally solve the model.

We used a portfolio of 3 admissible heuristics:

- $h_0$ assigns 0 to all states and serves as a baseline for the assessing the value of more informative heuristics.
- $h_{0^+}$ assigns 1 to all non-goal states and 0 otherwise.
- $h_{MinMin}$ solves all outcome determinization using the zero heuristic (Bonet and Geffner 2005).

Each problem was tested on a Intel(R) Xeon(R) CPU X5690 machine with a budget of 1, 2 and 3. Design actions were assigned a cost of $10^{-4}$, and convergence error bound of LAO* set to $10^{-6}$. Each run had a 30 minutes time limit.

**Results** Separated by domain and budget, Table 2 summarizes the number of solved instances (*solved*) and average percentage of expected cost reduction over instances solved (*reduc*). In all domains, complexity brought by increased budget reduces the number of solved instances, while the actual reduction varies among domains. As for solution improvement, all domains show an improvement of $15\%$ to $54\%$.

Table 3 compares solution performance. Each row represents a solver and heuristic pair. Results are separated by domain and budget, depicting the average running time for problems solved by all approaches for a given budget and the number of instances solved in parenthesis. The dominating approach for each row (indicating a domain and budget) is emphasized in bold. In all cases, the use of informed search outperformed the exhaustive approach.

| | BOX | | | BLOCKS | | | EX. BLOCKS | | | TRIANGLE TIRE | | | ELEVATORS | | | VACCUM | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | B=1 | B=2 | B=3 | B=1 | B=2 | B=3 | B=1 | B=2 | B=3 | B=1 | B=2 | B=3 | B=1 | B=2 | B=3 | B=1 | B=2 | B=3 |
| Ex-$h_0$ | 158.4(8) | 264.7(7) | 238.5(4) | 50.5(6) | 28.0(4) | 348.9(2) | 69.4(9) | 161.7 (9) | 250.7 (9) | 32.9(9) | 55.2(7) | 270.3(6) | 300.4 (8) | 361.8 (5) | na | 0.15(9) | 3.6(9) | na |
| Ex-$h_{0+}$ | 159.0(8) | 264.9(7) | 236.5(4) | 50.5(6) | 28.2(4) | 347.3(2) | 70.2(9) | 170.9(9) | 265.9(9) | 32.9(9) | 55.4(7) | 136.5(6) | 299.6(8) | 360.9(5) | na | 0.16(9) | 3.27(9) | na |
| Ex-$h_{MinMin}$ | 158.9(8) | 267.8(7) | 235.6 (4) | 50.8(6) | 28.0(4) | 348.2(2) | 69.9 (9) | 168.1(9) | 292.2(9) | 33.1(9) | 55(7) | 258.4(6) | 301.6(8) | 366.2 (5) | na | 0.152(9) | 3.44(9) | na |
| DC-$h_0$ | 163.9(8) | 270.6(7) | 241.5(4) | 50.7(6) | 28.2(4) | 354.5(2) | 68.4(9) | 153.1(9) | 252.5(9) | 33.3(9) | 55.5(7) | 269.7(6) | 301.9(8) | 363.4(5) | na | 0.17(9) | 3.25(9) | na |
| DC-$h_{0+}$ | 70.7(8) | 92.1(8) | 73.5(4) | 41.7(6) | 17.4(4) | 194.6(3) | 38.7(9) | 88.2(9) | 134.9(9) | 30.2(9) | 51.1(8) | **136.5**(6) | 236.2(9) | 261(5) | 1504.6(1) | 0.099(9) | 2.13(9) | na |
| DC-$h_{MinMin}$ | 221.4 (8) | 332.7(7) | 271.7(4) | 77.1(6) | 36.4(3) | 363.5(2) | **6.7(10)** | **30.2**(10) | **88.8(8)** | 36.8(9) | 88.8(8) | 258.4(6) | 192.6(9) | 243.89(5) | 1117.4(1) | 0.15(9) | 2.49(9) | na |
| BFD-$h_0$ | 157.4(8) | 260.8(7) | 234.3(4) | 50.3(6) | 28(4) | 352.2(2) | 69.5(9) | 153.9(9) | 285.9(9) | 33(9) | 55(7) | 267.6(6) | 302.6(8) | 360.8(5) | na | 0.15(9) | 3.39(9) | na |
| BFD-$h_{0+}$ | **68.2**(8) | **88**(8) | **70.2** (7) | **41.6** (6) | **17.2**(4) | **118.2**(3) | 40.3(9) | 85.6(9) | 160.9(9) | **29.5**(9) | **50.9**(8) | 188.3(6) | 238.3(9) | 258.6(5) | 1465.8(1) | **0.096(9)** | **2.021(9)** | na |
| BFD-$h_{MinMin}$ | 216.4(8) | 325.3(7) | 265.94(4) | 74.4(6) | 35.4(3) | 354.8(2) | 60.3(9) | 135(9) | 237.4(9) | 36.9(9) | 89(8) | 256.2(6) | **176.6**(9) | **231.2**(5) | **1042.5**(1) | 0.13(9) | 2.61(9) | na |

Table 3: Running time and number of instances solved for optimal solvers

## Approximate solutions

**Setup** For approximate solutions we used a solver based on an MDP reduction approach that creates simplified MDPs that account for the full set of probabilistic outcomes a bounded number of times (for each execution history), and treat the problem as deterministic afterwards (Pineda and Zilberstein 2014). The deterministic problems were solved using the FF classical planner (Hoffmann and Nebel 2001). Because this is a replanning approach, we used Monte-Carlo simulations to evaluate the policies' probability of reaching a goal state and its expected cost. In particular, we gave the planner 20 minutes to solve each problem 50 times. We used the first 10 instances of each competition domain mentioned above, excluding Box World, due to limitations of the planner. For the VACUUM domain we generated ten configurations of up to $5 \times 7$ grid size rooms, based on Figure 1.

**Results** Table 4 reports three measures (per budget): the number of problems completed within allocated time (*solved*), improved probability of reaching a goal of the resulting policies with respect to the policies obtained without design ($\delta P_s$), and the average percentage of reduction in expected cost after redesign (*reduc*) ($\delta P_s$ and *reduc* are averaged only over problems solved 50 times when using both the original and modified model). In general, we observed that redesign enables either improvement in expected cost or in probability of successes (and sometimes both), across all budgets. For BLOCK and EX-BLOCK, a budget of 2 yielded best results, while for ELEVATOR, TIRE, and VACUUM a budget of 3 was better. However, the increased difficulty of the compiled problem resulted sometimes in a lower number of solved problems (*e.g.*, solving only 3 problems on TIRE with budget of 3). Nevertheless, these results demonstrate the feasibility of obtaining good solutions when compromising optimality.

## Discussion

For all examined domains, results indicate the benefit of using heuristic search over an exhaustive search in the modification space. However, the dominating heuristic approach varied between domains, and for TIRE also between budget allocation. Investigating the reasons for this variance, we note a key difference between BFD and DC. While DC applies modifications to the original model, BFD uses the *simplified-environment* heuristic that applies them to a simplified model. Poor performance of BFD can be due to either the minor effect applied simplifications have on the computational complexity or to an exaggerated effect that may limit the informative value of heuristic estimations. In particular, this could happen due to the overlap between the design process and the simplification. To illustrate, by applying the all outcome determinization to the Vacuum domain depicted in Example 1, we ignore the undesired outcome of slipping. Consequently, the heuristic completely overlooks the value of adding high-friction tiles, while providing informative estimations to the value of (re)moving furniture. This observations may be used to account for the poor performance of BFD with EX-BLOCKS, where simplification via the delete relaxation ignores the possibility of blocks exploding and overlooks the value of the proposed modifications. Therefore, estimations of BFD may be improved by developing a heuristic that uses the aggregation of several estimations. Also, when the order of application is immaterial, a closed list may be used for examined sets in the BFD approach but not with DC. Finally, a combination of relaxation approaches may enhance performance of sub-optimal solvers.

| | $B=1$ | | | $B=2$ | | | $B=3$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | solved | $\delta P_s$ | reduc | solved | $\delta P_s$ | reduc | solved | $\delta P_s$ | reduc |
| BLOCK | 8 | 0 | 19.1 | 8 | 0 | 21.2 | 8 | 0 | 18.6 |
| EX-BLOCK | 10 | 0.42 | 0 | 10 | 0.50 | 0 | 10 | 0.41 | 0 |
| TIRE | 7 | 0 | 6.98 | 7 | 0 | 17.9 | 3 | 0 | 33 |
| ELEVATOR | 10 | -0.33 | 25 | 10 | 0.1 | 30 | 10 | 0.1 | 38.3 |
| VACUUM | 10 | 0.2 | 8.12 | 10 | 0.2 | 4.72 | 10 | 0.3 | 9.72 |

Table 4: Utility improvement for sub-optimal solver

## Related Work

Environment design (Zhang, Chen, and Parkes 2009) provides a framework for an interested party (system) to seek an optimal way to modify an environment to maximize some utility. Among the many ways to instantiate the general model, *policy teaching* (Zhang and Parkes 2008; Zhang, Chen, and Parkes 2009) enables a system to modify the reward function of a stochastic agent to entice it to follow specific policies. We focus on a different special case where the system is altruistic and redesigns the environment in order to maximize agent utility. The techniques used for solving the policy teaching do not apply to our setting, which supports arbitrary modifications.

The *DesignComp* compilation is inspired by the technique of Göbelbecker et al. (2010) of coming up with good excuses for why there is no solution to a planning problem. Our compilation extends the original approach in four directions. First, we move from a deterministic environment to a stochastic one. Second, we maximize agent utility rather than only moving from unsolvable to solvable. Third, we embed support of a design budget. Finally, we support arbitrary modification alternatives including modi-

fications specific to stochastic settings as well as all those suggested for deterministic settings (Herzig et al. 2014; Menezes, de Barros, and do Lago Pereira 2012).

## Conclusions

We presented the ER-UMD framework for maximizing agent utility by the off-line design of stochastic environments. We presented two solution approaches; a compilation-based method that embeds design into the definition of a planning problem and an informed heuristic search in the modification space, for which we provided an admissible heuristic. Our empirical evaluation supports the feasibility of the approaches and shows substantial utility gain on all evaluated domains.

In future work, we will explore creating tailored heuristics to improve planner performance. Also, we will extend the model to deal with partial observability using POMDPs, as well as automatically finding possible modifications, similarly to (Göbelbecker et al. 2010). In addition, we plan to extend the offline design paradigm, by accounting for online design that can be dynamically applied to a model.

## Acknowledgements

The work was partially supported by the NSF grant number IIS-1405550.

## References

Bellman, R. 1957. A markovian decision process. *Indiana University Mathematics Journal* 6:679–684.

Bertsekas, D. P. 1995. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA.

Bonet, B., and Geffner, H. 2005. mgpt: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research(JAIR)* 24:933–944.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research(JAIR)* 13:227–303.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. *Cognitive Robotics* 10081.

Herzig, A.; Menezes, V.; de Barros, L. N.; and Wassermann, R. 2014. On the revision of planning tasks. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI)*.

Hoffmann, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Holte, R. C.; Perez, M.; Zimmer, R.; and MacDonald, A. J. 1995. Hierarchical a*. In *Symposium on Abstraction, Reformulation, and Approximation*.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Keren, S.; Gal, A.; and Karpas, E. 2015. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI)*.

Mausam, A. K. 2012. Planning with markov decision processes: an ai perspective. *Morgan & Claypool Publishers*.

Menezes, M. V.; de Barros, L. N.; and do Lago Pereira, S. 2012. Planning task validation. In *Proceedings of the International Conference on Automated Planning and Scheduling Workshop on Scheduling and Planning Applications (SPARK-ICAPS)*, 48–55.

Nllsson, N. J. 1980. Principles of artificial intelligence. *TiogaSpringer Verlag. Palo Alto. California*.

Pineda, L., and Zilberstein, S. 2014. Planning under uncertainty using reduced models: Revisiting determinization. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Wayllace, C.; Hou, P.; Yeoh, W.; and Son, T. C. 2016. Goal recognition design with stochastic agent action outcomes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 7, 352–359.

Younes, H. L., and Littman, M. L. 2004. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proceedings of the International Planning Competition (IPC)*.

Zhang, H., and Parkes, D. C. 2008. Value-based policy teaching with active indirect elicitation. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI)*.

Zhang, H.; Chen, Y.; and Parkes, D. C. 2009. A general approach to environment design with one agent. *Morgan Kaufmann Publishers Inc.*