Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Efficient Heuristic Search for Optimal Environment Redesign

Sarah Keren,† Luis Pineda,‡ Avigdor Gal,‡ Erez Karpas,‡ Shlomo Zilberstein‡

†Harvard University, School of Engineering and Applied Sciences
‡Technion–Israel Institute of Technology
‡College of Information and Computer Sciences, University of Massachusetts Amherst
skeren@seas.harvard.edu, lpineda@cs.umass.edu*

Abstract

Given an environment, the utility measure of the agents acting within it, a set of possible environment modifications, and a description of design constraints, the objective of *equi-reward utility maximizing design* (ER-UMD) is to find a valid sequence of modifications to apply to the environment in order to maximize agent utility. To efficiently traverse the typically large space of possible design options, we use heuristic search and propose new heuristics, which relax the design process; instead of computing the value achieved by a single modification, we use a *dominating* modification guaranteed to be at least as beneficial. The proposed technique enables heuristic caching for similar nodes thereby saving computational overhead. We specify sufficient conditions under which our approach is guaranteed to produce admissible estimates, and describe a range of models that comply with these requirements. Also, for models with lifted representations of environment modifications, we provide simple methods to automatically generate dominating modifications. We evaluate our approach on a range of stochastic settings for which our heuristic is admissible. We demonstrate its efficiency by comparing it to a previously suggested heuristic, that employs a relaxation of the environment, and to a compilation from ER-UMD to planning.

Introduction

Equi-reward utility maximizing design (ER-UMD) (Keren et al. 2017) involves redesigning environments in order to maximize agent performance. The input of an ER-UMD problem consists of a description of a stochastic environment, a utility measure of the agents acting within it, a set of design constraints, and the available environment modifications (hereon, modifications), that represent the possible ways to modify and redesign the environment. The objective is to find a sequence (or set when modifications are unordered) of modifications to apply to the environment to maximize agent utility. The design process is viewed as a search in the often exponential space of possible modification sequences, motivating the use of heuristic estimations to guide the search.

ER-UMD is relevant to a variety of complex systems that can be modified for improved utility. Given a wide range of environment modifications, policy makers need to choose a sequence of applied modifications that yields maximal benefit, while respecting various design constraints, such as a design budget. When it is impractical to exhaustively explore the large space of design options, heuristic search can be applied to increase efficiency, using heuristics to estimate the value of a given modification. Specifically, when optimal solutions are sought, admissible estimations are those guaranteed to over-estimate the value of a modification.

Typically, it may be hard or impossible to accurately predict the effect and potential benefit of each modification. Moreover, in many cases, a large portion of the possible modifications has little or no effect on utility. Computing the heuristic value for each such modification is wasteful. In this work we therefore present the *simplified-design* heuristic, which relaxes the modification process by mapping each modification that is expanded during the search to a modification that *dominates* it, *i.e.*, a modification guaranteed to yield a value at least as high, and use its value as an estimation of the value of the original modification.

To generate dominating modifications, we propose two approaches, namely *modification relaxation* and *padding*. Modification relaxation consists of applying a hypothetical modification whose effect is potentially easier to compute than the original modification. Padding appends to the examined modification additional modifications. The computed values of padded modifications are cached. When a modification is mapped to a previously encountered padded modification, the cached value is reused. Both approaches can be combined with the potential benefit lying in the ability to avoid redundant computations of irrelevant sets of modifications, those that do not affect the agent's expected utility.

For models with lifted modification representations, we provide a simple way to automatically generate dominating modifications. We then specify sufficient conditions under which this approach is guaranteed to produce admissible heuristics, *i.e.*, heuristics that over-estimate the value of the original modification. In addition, we formulate and implement a family of models that comply with these requirements. We compare the efficiency of our proposed approach with that of a previously suggested heuristic that employs an environment relaxation and with a compilation from ER-UMD to planning.

Our modification padding technique, which generates

*Last three authors email addresses: avigal@ie.technion.ac.il, karpase@technion.ac.il, shlomo@cs.umass.edu

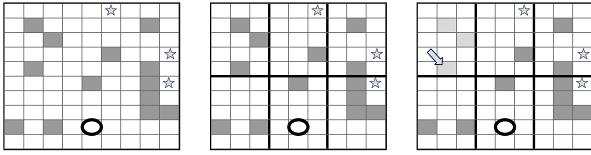

Figure 1: An example ER-UMD problem

dominating modifications, is inspired by pattern database (PDB) heuristic approaches, originally developed for planning problems (Culberson and Schaeffer 1998; Haslum et al. 2007; Edelkamp 2006). PDBs are abstraction heuristics that ignore some aspects of a search problem (the *pattern*) in order to create a problem that can be optimally solved efficiently. The key difference between padding and pattern database heuristics is that the former does not necessarily yield an easier-to-solve model. Instead, it potentially avoids redundant computations of irrelevant modification sets, those that do not affect the agent's expected utility.

Example 1 *To illustrate our* simplified-design *heuristic, consider Figure 1(left) where an adaptation of the Vacuum cleaning robot domain suggested by Keren et al. (2017) is portrayed. The setting includes a robot (depicted by a black circle) that needs to collect, as quickly as possible, pieces of dirt (depicted by stars) scattered in the room. The robot needs to navigate around the furniture in the room, depicted by shaded cells. Accounting for uncertainty, the robot may slip when moving, ending up in a different location than intended. To facilitate the robot's task, the environment can be modified by removing furniture, or by placing high friction tiles to reduce the probability of slipping. The design process is constrained by a design budget, limiting the number of allowed modifications.*

The simplified-design *heuristic is implemented by partitioning the environment into* zones *(Figure 1(center)). To heuristically evaluate the impact of removing the piece of furniture indicated by the arrow in Figure 1(right), we remove all furniture from its entire zone and use this value as an (over) estimation of the single modification. When considering the removal of another piece of furniture in the same zone, the already computed value is reused.*

While the example above provides a simple illustration of the ideas we present in this work, we view them as relevant to various real-world applications. Consider, for example, a large road network, used by both autonomous and manned vehicles, where a limited number of costly sensors can be distributed in specific locations to enhance the ability to perceive the current road conditions, such as the position of human pedestrians, road blockage, *etc*. In such settings, sensors are often noisy and inaccurate, making it hard to predict the exact value (*i.e.*, benefit) of each sensor placement. In addition, modifying the sensor distribution in areas with light traffic may have little effect on the overall utility. In such settings, the value of a sensor can be over-estimated using the value of a set of noiseless sensors. This value can be reused for the estimation of sensors in adjacent locations. This is useful in particular when the application of the superset sensor is useless, and the computation effort invested in more accurate overestimations would be redundant.

The main contributions of this work are threefold. First, we propose a new class of heuristics for ER-UMD, called *simplified-design*. Specifically, we suggest caching heuristic values as a way to avoid many repeated computations and increase efficiency. Second, we identify conditions under which this class of heuristics is admissible. Finally, we describe a concrete procedure to automatically generate such heuristics. Our empirical evaluation shows the benefit of our proposed approach on a variety of domains.

In the remaining of the paper we first overview the ER-UMD framework, and then describe our novel techniques for solving the ER-UMD problem. Our empirical evaluation is followed by a description of related work and concluding remarks.

Background: Equi-Reward Utility Maximizing Design as Heuristic Search

The *equi-reward utility maximizing design* (ER-UMD) problem, recently suggested in (Keren et al. 2017), takes as input an environment with stochastic action outcomes, a utility measure of the agents that act in it, a set of allowed modifications, and a set of design constraints. The aim is to find an optimal sequence of modifications to apply to the environment for maximizing agent utility (alternatively, minimizing expected cost) under the specified constraints.

The ER-UMD framework considers stochastic environments defined by the quadruple $\epsilon = \langle S, A, f, s_0 \rangle$ with a set of states S, a set of actions A, a stochastic transition function $f : S \times A \times S \rightarrow [0, 1]$ specifying the probability $f(s, a, s')$ of reaching state s' after applying action a in $s \in S$, and an initial state $s_0 \in S$. We let $\mathcal{E}$, $\mathcal{S}_\mathcal{E}$ and $\mathcal{A}_\mathcal{E}$ denote the set of all environments, states and actions, respectively.

An ER-UMD model is a tuple $\omega = \langle \epsilon^0, \mathcal{R}, \gamma, \mathcal{F}, \mathbf{\Delta}, \Phi \rangle$ where, $\epsilon^0 \in \mathcal{E}$ is an initial environment, $\mathcal{R} : \mathcal{S}_\mathcal{E} \times \mathcal{A}_\mathcal{E} \times \mathcal{S}_\mathcal{E} \rightarrow \mathbb{R}$ 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, and γ is a discount factor in $(0, 1]$, representing the deprecation of agent rewards over time. Letting $\mathbf{\Delta}^u$ represent the set of all modifications, $\mathcal{F} : \mathcal{E} \times \mathbf{\Delta}^u \rightarrow \mathcal{E}$ is a deterministic modification transition function, specifying the result of applying a single modification to an environment in ω. In the following, we assume any modification is applicable to any environment, and invalid modifications leave the model unchanged. The set $\mathbf{\Delta} \subseteq \mathbf{\Delta}^u$ contains the atomic modifications that are available in ω. A modification sequence is an ordered set of modifications $\vec{\delta} = \langle \delta_1, \ldots, \delta_n \rangle$ s.t. $\delta_i \in \mathbf{\Delta}^u$ and $\vec{\mathbf{\Delta}}$ is the set of all such sequences. Equivalently, the set $\vec{\mathbf{\Delta}}_\omega$ is the set of sequences available in ω, which includes the empty sequence $\vec{\delta}_\emptyset$. Finally, $\Phi : \mathcal{E} \times \vec{\mathbf{\Delta}} \rightarrow \{0, 1\}$ represents the design constraints and specifies allowed modification sequences in an environment.

An environment $\epsilon \in \mathcal{E}$, discount factor γ, and reward function $\mathcal{R}$ represent an infinite horizon discounted reward Markov decision process (MDP) (Bertsekas 1995)

$\langle S, A, f, s_0, \mathcal{R}, \gamma \rangle$. We assume agents are optimal, *i.e.*, they follow a policy that maximizes their utility, expressed as their expected accumulated reward.

Given an ER-UMD problem ω, our objective is to find a legal modification sequence that, when applied to the initial environment ϵ^0, maximizes agent utility. Keren et al. (2017) propose to view the design process as a search in the space of modification sequences. Our search is therefore a search in the space $\vec{\mathbf{\Delta}}_\omega$ of possible modifications sequences in ω. A search node is a modification sequence $\vec{\delta} \in \vec{\mathbf{\Delta}}_\omega$ applied to ϵ^0. Accordingly, the value of each node $\vec{\delta}$ is denoted by $\mathcal{V}^*_\omega(\vec{\delta})$, which represents the maximal agent utility achievable via redesign from node $\vec{\delta}$.

Note that we use $\vec{\delta}$ to characterize a node rather then the modified environment it imposes, since we want to support preferences over modification sequences that achieve the same utility (by leading to the same environment). To represent the utility achievable via design, a node needs to represent the value of the possible modifications that can be applied to its underlying environment, reflected by its corresponding modification sequence.

In (Keren et al. 2017) two methods are suggested for searching the space of modification sequences. The first, referred to as *DesignComp*, embeds the offline design stage into the definition of the agent's planning problem (*i.e.*, MDP description), which can be solved by any off-the-shelf MDP solver. The second approach, namely the Best First Design (BFD) algorithm, applies a best first search (Pearl 1984) in the space of modification sequences (see Figure 2 for illustration). The root node represents the initial environment (and empty modification sequence $\vec{\delta}_\emptyset$), and each successor node is generated by applying an available modification. The internal *design nodes* represent the design process and are estimated using a heuristic function. The heuristic is admissible if it never underestimates the maximal utility achievable via redesign from a node. The leaf nodes of the search tree, dubbed *execution nodes*, represent the redesigned stochastic environments in which agents act. One execution node is generated for each modification sequence encountered during the search, and is evaluated by the expected agent utility in the modified environment. BFD iteratively explores the currently most promising node, halting when an execution node is deemed best. When using an admissible heuristic, BFD is guaranteed to return an optimal solution. To produce admissible estimation of the value of a modification sequence efficiently, the *simplified-environment* heuristic was proposed, relaxing the environment using relaxation approaches from the literature (*e.g.*, delete relaxation that ignores the negative outcomes of an action (Bonet, Loerincs, and Geffner 1997)), before evaluating a modification on the relaxed environment.

The *simplified-design* Heuristic

To estimate the value of a modification, we *relax* the design process by mapping the modification to a modification that *dominates* it, meaning it achieves a utility at least as high as the original modification's utility. This approach can be exploited in two ways. First, if the value of the dominating modification is easier to compute, it can be used to estimate the value of the original modification. In addition, we can cache the computed values and reuse them for each encountered node (and corresponding modification) that is dominated by the same relaxed modification.

We start with a definition of the *simplified-design* heuristic. Then, we characterize ER-UMD settings where modification relaxation is easy to implement, and in which our approach is guaranteed to yield admissible heuristic, *i.e.*, overestimations of the expected value of the applied modifications.

Recall that $\vec{\mathbf{\Delta}}_\omega$ represents all possible modification sequences in ω. In addition, we let $\vec{\delta} \cdot \delta$ represent the modification sequence that results from appending δ to $\vec{\delta}$ and define a dominating modification as follows.

Definition 1 (dominating modification) *Given an ER-UMD model $\omega = \langle \epsilon^0, \mathcal{R}, \gamma, \mathbf{\Delta}, \mathcal{F}, \Phi \rangle$, a modification δ'* dominates *modification δ in ω if for every $\vec{\delta} \in \vec{\mathbf{\Delta}}_\omega$*

$$\mathcal{V}^*_\omega(\vec{\delta} \cdot \delta) \leq \mathcal{V}^*_\omega(\vec{\delta} \cdot \delta')$$

For each node $\vec{\delta}$, the *simplified-design* heuristic, denoted by h^{simdes}, estimates the value of $\vec{\delta} \cdot \delta$ by the value of appending to $\vec{\delta}$ a modification δ' that dominates δ.

$$h^{simdes}(\vec{\delta} \cdot \delta) \stackrel{def}{=} \mathcal{V}^*_\omega(\vec{\delta} \cdot \delta') \tag{1}$$

Lemma 1 *h^{simdes} is admissible in any ER-UMD model ω.*

Proof: Immediate from the definition of dominance. ■

Admissibility of dominating modifications

The *simplified-design* heuristic creates dominating modifications using two main methods, namely *relaxation* and *padding*, to be discussed next.

Modification relaxation uses the dominance relation between modifications (Definition 1) to generate modifications guaranteed to be at least as beneficial as the original ones. Applying a relaxed modification is guaranteed to produce admissible estimates since, by definition, the relaxed modification is guaranteed to return a value that is no lower than the original modification. It is worth noting that the relaxed modification is not necessarily applicable in reality, yet may result in a model for which utility is calculated more efficiently.

In Example 1, we can estimate the value of applying a high friction tile that reduces the probability of slipping from 50% to 10% by using the value of applying a relaxed hypothetical modification that reduces the probability of slipping to 0. Ignoring the probabilistic nature of the modified environment potentially reduces the computational overhead of the actual setting.

Modification padding is a dominating modification that involves integrating the explored modification into a sequence of modifications.

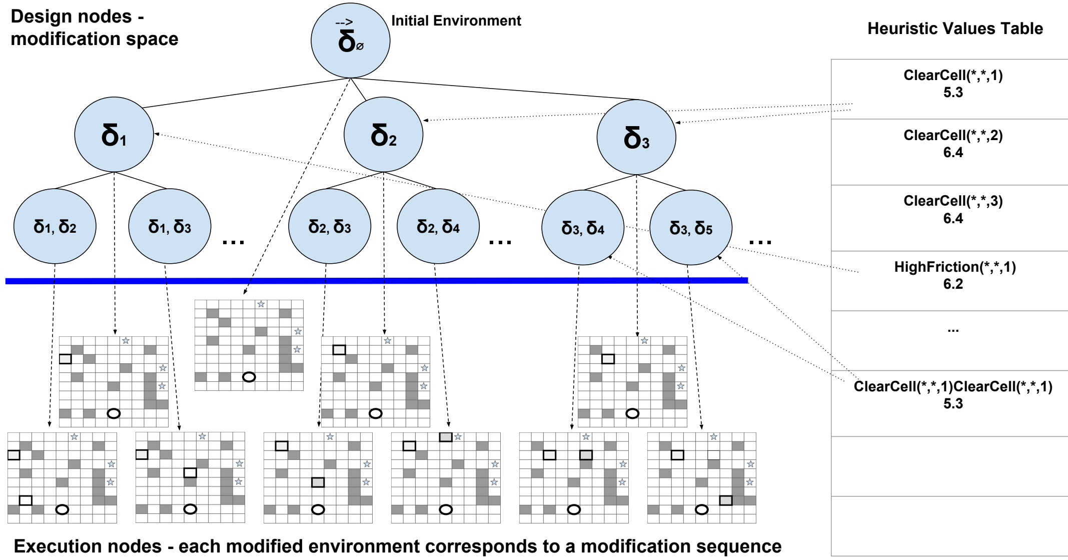

Figure 2: An illustration of the design process using cached values

Definition 2 (padded modification) $\hat{\delta} = \langle \delta_1, \ldots, \delta_n \rangle$, *is a* padded modification *of modification* δ, *if* $\delta_j \in \mathbf{\Delta}^u$ *for all* $1 \leq j \leq n$ *and* $\exists i \ 1 \leq i \leq n$ *s.t.* $\delta = \delta_i$.

Note that the modification sequences that comprise the padded modifications are distinct from the modification sequences of our state space $\vec{\mathbf{\Delta}}_\omega$. Padded modifications are only used to extract heuristic estimations. Also note that, as opposed to modification relaxation, the benefit of applying modification padding does not lie in the ability to create models that are necessarily easier to solve. Instead, this approach potentially reduces the computational effort of the search by avoiding redundant evaluations of modifications that affect aspects of the model that have little or no impact on the agent expected utility. Particularly, we can cache values of previously computed nodes (and their padded sequences) and reuse theses values for 'similar' nodes that represent modifications that are mapped to the same padded sequence.

In Example 1, modification padding can be implemented by estimating the value of removing a single piece of furniture, using the value of removing all pieces of furniture from an entire zone (depicted in Figure 1(right)). Similarly, the value of adding a single high-friction tile can be estimated by the value of adding a set of tiles to the entire zone to which the cell belongs.

The design process of our example is depicted in Figure 2, with the left part of the image representing the search in the space of modification sequences. On the top left, we have the design nodes of the BFD search tree, where each node represents a legal modification sequence. Each node corresponds to a modified environment (execution node) at the bottom of the image. To estimate the value of a modification, padding is applied, and the computed value is stored in a table (on the right). When expanding a modification sequence that is mapped to the same padded sequence, the precomputed value is reused.

Naturally, both relaxation and padding techniques can be combined by first applying a modification relaxation and then padding it with a sequence of additional modifications. We call this a *relaxed padded modification*, for which the definition is an immediate extension of definitions 1 and 2. Note that modification relaxation is a special case of relaxed modification padding when the sequence appended to the modification is empty. Similarly, modification padding is also a special case where a modification δ is mapped to itself and then padded.

While using modification relaxation always yields admissible estimates, padding sequences may under-estimate the value of a modification. This can happen, for example, when modifications can cancel the effect of other modifications (e.g., modifications that add objects to a room cancel the effect of those that remove them). We show that when an ER-UMD model is both *independent* (modification sequences applied in any order yield the same result) and *monotonic* (no modifications can reduce agent utility), sequence padding never under-estimates a modification and can therefore be used to extract admissible estimates. We give an exact characterization of both notions below.

Definition 3 (monotonic model) *An ER-UMD model* ω *is* monotonic *if for every modification* $\delta \in \mathbf{\Delta}$ *and modification sequence* $\vec{\delta} \in \vec{\mathbf{\Delta}}_\omega$

$$\mathcal{V}^*_\omega(\vec{\delta}) \leq \mathcal{V}^*_\omega(\vec{\delta} \cdot \delta)$$

Definition 4 (independent model) *An ER-UMD model* ω *is* independent *if for any modification sequence* $\vec{\delta} \in \vec{\mathbf{\Delta}}_\omega$, *and modification sequence* $\vec{\delta}'$ *that is a permutation of* $\vec{\delta}$,

$$\mathcal{V}^*_\omega(\vec{\delta}) = \mathcal{V}^*_\omega(\vec{\delta}')$$

Lemma 2 *Given a monotonic independent ER-UMD model ω, a modification δ and a relaxed padded modification $\hat{\delta}$, for every node $\vec{\delta} \in \vec{\mathbf{\Delta}}_\omega$*

$$\mathcal{V}^*_\omega(\vec{\delta} \cdot \delta) \leq \mathcal{V}^*_\omega(\vec{\delta} \cdot \hat{\delta})$$

Proof: (sketch) Since the model is independent, we can apply the modifications in any order. In particular, we can first apply $\delta_i \in \hat{\delta}$ that dominates δ, and get a value that overestimates $\mathcal{V}^*_\omega(\vec{\delta} \cdot \delta)$. Since the model is monotonic, applying the additional modifications in the sequence is guaranteed to yield a value at least as high as $\mathcal{V}^*_\omega(\vec{\delta} \cdot \delta)$. ■

Corollary 1 *The* simplified-design *heuristic is admissible in any monotonic and independent ER-UMD model ω.*

The proof for Corollary 1 is immediate from Lemma 2.

A direct implication of Corollary 1 is in providing conditions under which the *simplified-design* heuristic can be used by an optimal heuristic search algorithm, and BFD (Keren et al. 2017) in particular, to produce an optimal solution to a ER-UMD problem.

Modifications for Independent Monotonic ER-UMD models

After specifying the conditions under which our approach is guaranteed to yield admissible estimations, we now characterize ER-UMD models that comply with these requirements. Specifically, we characterize monotonic and independent models where modification padding can be used to produce admissible estimates.

For this purpose, we define *action addition* modifications that add applicable actions to some states of the model. We then show that ER-UMD models that allow only action addition modifications are both independent and monotonic.

To formally define action addition modifications, we let $app(s, \epsilon) \subseteq A$ represent the actions applicable in state s of environment ϵ.

Definition 5 (action addition modification) *Modification δ is an* action addition *modification (ADM) in ER-UMD model ω, if for any environment $\epsilon \in \mathcal{E}$, $\mathcal{F}(\epsilon, \delta)$ is identical to ϵ except that for every state $s \in S$ there exists a (possible empty) set of actions $A_{s,\delta}$ s.t. $app(s, \mathcal{F}(\epsilon, \delta)) = app(s, \epsilon) \cup A_{s,\delta}$.*

In Example 1, action addition is exemplified both by removing furniture, implemented by enabling move actions between previously disconnected states, and by placing high-friction tiles, implemented by adding to the model actions with a reduced probability of slipping.

Lemma 3 *An ER-UMD model with only action addition modifications is independent and monotonic.*

Proof: (sketch) Every action applicable in any state of the original model is applicable in the modified one. The expected utility of the initial state cannot be reduced as a result of applying a modification and is therefore monotonic. Following Definition 5, any two modifications $\delta, \delta' \in \mathbf{\Delta}$ can be applied in any order to yield the same set of applicable actions. This can be applied for any pair of modifications in a sequence, indicating that the model is independent. ■

It is worth noting that all modifications used in (Keren et al. 2017), including those implemented as initial state modifications, were in fact action addition modifications, since they changed the initial state in such a way that enabled more actions in some of the states reachable from the initial state. As demonstrated above, removing a piece of furniture in Example 1 can be modeled as enabling the movement to a previously occupied cell. In general, however, not all initial state modifications are monotonic. For example, when we remove from the initial state a fact that is a precondition of an action or add a fact that is a negative precondition, we may cause an action to become non-applicable and reduce utility.

Automatic Dominating Modification Generation

Having characterized ER-UMD models where our *simplified-design* approach, implemented via padding and modification relaxation, is guaranteed to produce admissible estimations, we now show two examples of how dominating modifications can be automatically generated.

First, to characterize models where modification padding is easily implemented, we focus our attention on *lifted* modifications that represent a set of parameters whose (grounded) instantiations define single modifications. Each lifted modification $\delta(p_1, \ldots, p_n)$ is characterized by a set of parameters $p_1, \ldots, p_n$ and a set of valid values $dom(p_i)$ for each parameter p_i. A (grounded) modification $\delta(v_1, \ldots, v_n)$ is a valid assignment to all parameters s.t. $v_i \in dom(p_i)$.

For lifted modifications, modification padding can be implemented using *parameterized padding*, by mapping a grounded modification to a sequence of modifications with the same values on a set of lifted parameters. In Example 1, $ClearCell(x, y, z)$ stands for the lifted representation of furniture removal modifications, where parameter z is the zone and x and y denote the cell coordinates within the zone. The value of the grounded modification $ClearCell(1, 3, 1)$ can be (over)estimated by the value of applying the sequence $ClearCell(1, 1, 1)$, $ClearCell(1, 2, 1)$, $ClearCell(1, 3, 1)$, *etc.* This value is cached, so when modification $ClearCell(1, 2, 1)$ is examined, it is mapped to the same padded sequence, and the precomputed value can be reused. Specifically, in our empirical evaluation, we use the lifted PPDDL (Younes and Littman 2004) representation to implement parameterized padding. This lifted representation enables an automatic mapping of a modification to its corresponding padded sequence.

Whenever modifications involve changing the probability distribution of an action's outcome, a relaxation can be automatically generated by creating a separate action for each of the outcomes (known in the literature as *all outcome determinization (Yoon, Fern, and Givan 2007)*). Continuing with Example 1, for a modification that adds high friction tiles to

reduce the probability of slipping from 50% to 10%, applying all outcome determinization creates a hypothetical dominating modification by allowing an agent to choose between two deterministic actions, either slipping or not.

Empirical Evaluation

Our empirical evaluation is dedicated to measuring the effectiveness of the proposed *simplified-design* heuristic on a variety of independent monotonic ER-UMD models, comparing it to the previously suggested *DesignComp* compilation and *simplified-environment* heuristic (Keren et al. 2017). In addition to reaffirming the benefit of using heuristic search for ER-UMD, we show the efficiency gain brought by caching and reusing previously computed values. We also analyze the role of different heuristics used to solve the underlying MDPs, that correspond to the applied modifications and the modified environments.

We used six PPDDL (Younes and Littman 2004) domains, adapted from Keren et al. (2017) that include five stochastic shortest path MDPs with uniform action cost domains from the probabilistic tracks of the eighth International Planning Competition: Boxworld (IPPC08/BOX), Elevators (IPPC08/ELE), Blocks World (IPPC08/BLOCK), Exploding Blocks World (IPPC08/EX. BLOCK), and Triangle Tire (IPPC08/TIRE). In addition, we used the vacuum cleaning robot setting adapted from Keren et al. (2017) and described in Example 1 (VACUUM). It is worth noting that the VACUUM domain is tailored to test the ER-UMD setting and the ability to improve upon an initial design.

	deterministic - enabled actions	**probabilistic- increase success probability**
BOX	move truck	drive
ELEVATOR	add shaft (enable step in and out)	move
BLOCK	put block on table (enable stack)	picking up a block or tower
EX-BLOCK	as for Blocks World	as for Blocks World
TIRE	add spare tires (enabled tire change) add road (enabled move)	drive (no flat tire)
VACUUM	remove furniture (enable move)	move

Table 1: Applicable modifications per domain

In all domains, agent utility is expressed as expected cost and constraints as a design budget. For each domain, we extend the modifications described by Keren et al. (2017), by including at least two modifications among which at least one alters the initial state and the other, the probability distribution (see Table 1 for a detailed description of the modifications implemented for each domain). All modifications were implemented as action addition modifications (Definition 5). Accordingly, all the tested models are independent and monotonic, which means that all our generated estimations are admissible and therefore over-estimate the expected utility $\mathcal{V}^*_\omega(\vec{\delta})$ of the modified model, for any model ω and sequence $\vec{\delta}$.[1]

Setup Each instance was solved using the following 7 solution approaches:

- BFS - an exhaustive breadth first search in the space of modifications.
- *DesignComp* (DC)- a compilation of the design problem to a planning problem, which embeds the design into the domain description (Keren et al. 2017).
- BFD- Best First Design, a heuristic best first search in the deterministic space of modifications. For BFD, we examined five heuristic approaches, the first of which was presented by Keren et al. (2017) and the other four are variations of the relaxed modification heuristic approach proposed in this work.
  - **rel-env** the *simplified-environment* heuristic where node evaluation is done on a relaxed environment (implemented by ignoring the delete effects of actions).
  - **rel-mod** the *simplified-design* heuristic that estimates the value of a modification by a single dominating modification.
  - **rel-comb** the *simplified-design* heuristic that combines both approaches above and estimates the value of a modification by a single dominating modification in a relaxed environment.
  - **rel-proc** the *simplified-design* heuristic that estimates the modification value using parametrized padding (on the first parameter of a modification), caching, and reusing the computed values.
  - **rel-comb-proc** the *simplified-design* heuristic that combines rel-proc and rel-mod, using parameterized padding of relaxed modifications, caching, and reusing pre-computed values.

BFD was implemented as a deterministic best first search in the design space. To evaluate the execution nodes (the leaf nodes of the BFD search space), *i.e.,* the expected agent utility in the modified stochastic environments, we used LAO* (Hansen and Zilberstein 1998) with convergence error bound of $\epsilon = 10^{-6}$. LAO* was also used for solving DC (the compilation). For LAO*, we used two heuristics. The *Min-Min* heuristic (Bonet and Geffner 2005) (h_{MIN}) uses all outcome determinization with the zero heuristic. We also implemented the *bounded all-outcome determinization* (heuristic h_{BAOD}) as a depth-bounded BFS exploration of all outcome determinization (for each domain we examined bounds of 5, 10, 15 and 20). Both heuristics are admissible.

All evaluations were performed on an Intel(R) Xeon(R) CPU X5690 machine with a budget of 1 to 5. To implement the budget constraint we added a counter to ensure that the number of design actions does not exceed the budget. Each run had a 20 minutes time limit.

Results Tables 2-7 summarize the results acquired for each design budget (indicated by $B = i$), separated by domains and heuristics. For each budget, we ran all 7 solution approaches presented above, each with h_{BAOD} and h_{MIN}, a total of 14 runs for each setting. For clarity of presentation, we present for each domain the dominating LAO* heuristic results only (either h_{BAOD} or h_{MIN}), indicated in parentheses below the domain name. For all but the EX. BLOCK domain, h_{BAOD} was the dominating approach. The five bottom rows of each table represent the BFD approach, with the corresponding design heuristic used (*e.g.*, BFD-rel-mod represent the BFD search with rel-mod as the design heuristic).

[1] Our set of benchmarks, code, and ER-UMD problem generator can be found in https://github.com/sarah-keren/ER-UMD-2019

	Budget=1 ($\mathcal{V}^* = 5.5$)		Budget=2 ($\mathcal{V}^* = 5.2$)		Budget=3 ($\mathcal{V}^* = 5.0$)		Budget=4 ($\mathcal{V}^* = 4.6$)		Budget=5 ($\mathcal{V}^* = 4.1$)	
	sol	time	sol	time	sol	time	sol	time	sol	time
BFS	1	0.68	1	3.04	0.3	229.65	0	NA	0	NA
DC	1	1.89	1	3.14	0.3	425.99	0	NA	0	NA
BFD-rel-env	1	0.20	1	3.11	0.5	28.65	0.3	198.95	0.2	206.43
BFD-rel-mod	1	0.26	1	3.15	0.5	29.35	0.3	203.71	0.2	202.51
BFD-rel-comb	1	0.21	1	3.81	0.5	28.88	0.3	204.95	0.2	203.51
BFD-rel-proc	1	0.14	1	**2.84**	0.5	**5.79**	0.3	32.28	0.2	**38.67**
BFD-rel-comb-proc	1	**0.12**	1	3.03	0.5	6.10	0.3	**31.0**	0.2	39.09

Table 2: Vaccum
(h_{BAOD})

	Budget=1 ($\mathcal{V}^* = 7.0$)		Budget=2 ($\mathcal{V}^* = 5.3$)		Budget=3 ($\mathcal{V}^* = 4.9$)		Budget=4 ($\mathcal{V}^* = 4.1$)		Budget=5 ($\mathcal{V}^* = 3.7$)	
	sol	time	sol	time	sol	time	sol	time	sol	time
BFS	1	0.08	1	0.68	0.7	3.41	0.4	13.2	0.3	46.96
DC	1	0.06	1	0.05	0.9	0.42	0.6	3.76	0.4	15.83
BFD-rel-env	1	**0.01**	1	**0.03**	0.9	0.09	0.6	0.35	0.6	1.15
BFD-rel-mod	1	**0.01**	1	0.06	0.9	0.19	0.6	0.73	0.6	1.34
BFD-rel-comb	1	0.02	1	0.08	0.9	0.22	0.6	0.79	0.6	1.3
BFD-rel-proc	1	0.02	1	0.04	0.9	**0.05**	0.7	**0.10**	0.7	0.16
BFD-rel-comb-proc	1	0.02	1	0.05	0.9	0.06	0.7	0.11	0.7	**0.15**

Table 3: Triangle Tire (TIRE)
(h_{BAOD})

	Budget=1 ($\mathcal{V}^* = 3.1$)		Budget=2 ($\mathcal{V}^* = 2.8$)		Budget=3 ($\mathcal{V}^* = 2.2$)		Budget=4 ($\mathcal{V}^* = 2.0$)		Budget=5 ($\mathcal{V}^* = NA$)	
	sol	time	sol	time	sol	time	sol	time	sol	time
BFS	1	0.19	1	2.92	1	32.09	0.8	367.87	0	NA
DC	1	**0.07**	1	1.12	1	14.72	1	214.43	0	NA
BFD-rel-env	1	0.19	1	3.67	1	43.89	0.8	493.36	0	NA
BFD-rel-mod	1	0.21	1	3.15	1	48.61	0.7	569.74	0	NA
BFD-rel-comb	1	0.25	1	2.93	1	51.58	0	NA	0	NA
BFD-rel-proc	1	0.19	1	0.59	1	14.28	1	171.44	0	NA
BFD-rel-comb-proc	1	0.18	1	**0.48**	1	**10.13**	1	**122.23**	0	NA

Table 4: Blocksworld (BLOCK)
(h_{BAOD})

	B=1 ($\mathcal{V}^* = 10.5$)		B=2 ($\mathcal{V}^* = 7.2$)		B=3 ($\mathcal{V}^* = 4.01$)		B=4 ($\mathcal{V}^* = 3.43$)		B=5 ($\mathcal{V}^* = NA$)	
	sol	time	sol	time	sol	time	sol	time	sol	time
BFS	1	2.04	1	66.24	0	NA	0	NA	0	NA
DC	1	0.86	1	21.38	0	NA	0	NA	0	NA
BFD-rel-env	1	0.33	1	7.82	0.9	98.59	0	NA	0	NA
BFD-rel-mod	1	0.58	1	12.31	0.9	124.52	0	NA	0	NA
BFD-rel-comb	1	0.66	1	23.28	0.9	148.25	0	NA	0	NA
BFD-rel-proc	1	0.32	1	**2.82**	0.9	**22.48**	0.3	**434.5**	0	NA
BFD-rel-comb-proc	1	**0.26**	1	3.34	0.9	26.01	0.3	492.34	0	NA

Table 5: Elevators (ELE)
(h_{BAOD})

For each design budget, the tables indicate $\mathcal{V}^*$ as the expected cost achieved via redesign, while NA indicates settings not solved by any approach (since all our solutions are optimal, the value is the same for all approaches that completed within the allocated time). In all domains, the effect of redesign is demonstrated by the reduction in expected cost as the budget increases. This is to be expected, given that all our examined models are monotonic and independent. For each combination of budget and solution approach, sol is the ratio of problem solved to completion and $time$ is the average running time in seconds for problems commonly solved by all methods (those that did not fail on all instances, a case which is indicated by NA). For each budget, the dominating approach is indicated in bold.

For most domains, the results clearly indicate the efficiency of solving ER-UMD using the BFD heuristic search approach as suggested by Keren et al. (2017). Even for settings where the DC approach, that includes the solution of single MDP that encapsulates the design modifications, outperforms all other approaches for small budgets, the benefit of the heuristic search and the design heuristics becomes apparent as the budget increases. The only exception is the BOX domain, where DC and the caching-based heuristic approaches achieve similar results (our caching-based approach is slightly faster, and neither approach solves problems for budget 3 and higher). We associate this to the savings in the backward propagation of updated values, performed at each iteration by LAO*. For DC, value updates of each node are propagated all the way up to the single root, while for BFD, the updates are done separately for each design node and its corresponding environment. The deeper and wider the search tree, the greater the savings.

Among the five heuristics used by BFD, the superiority of our caching based approaches, represented by BFD-rel-proc and BFD-rel-comb-proc at the two bottom rows of each table, is absolute. For all domains, the benefit of caching dominating modifications and reusing these values for similar modifications increases with the size of the examined problem (and the budget in particular). For bigger problems, our caching based methods not only solve the commonly solved problems more quickly, but solve more problems than any other approach (*e.g.*, for the ELE domain, our cashing approaches were the only ones to solve instances for budget of 4).

To investigate this trend further, we counted the number of nodes expanded at each run by counting the number of calls to the LAO* heuristic (h_{BAOD} or h_{MIN}). Among these nodes, we counted the number of design nodes, representing the different examined modification sequences. The results show that the number of calls to the LAO* heuristic is not necessarily positively correlated to the overall running time. In some instances, the number of calls to the heuristic may be smaller than for instances with a lower running time. However, in all cases, the caching approach examines far fewer design nodes.

Taking the BLOCK domain, for example, with the h_{BAOD} as the LAO* heuristic, for a budget of 1 (where DC is the dominating approach) the average total number of expanded nodes by the DC compilation is 902.3, among which 558.4 are design nodes (many of which are re-examined several times). The rel-comb-proc approach examines 2688.5, among which an average of 81.3 are design nodes. In contrast, for a budget of 4, for which the rel-comb-proc approach dominates all other approaches, DC examines 1163564.5 and 794577.4 of total and design nodes respectively, while the rel-comb-proc examines a slightly higher number of nodes (1283676.5), among which only 1920.5 are

	B=1 ($\mathcal{V}^* = 52.1$)		B=2 ($\mathcal{V}^* = 12.3$)		B=3 ($\mathcal{V}^* = 2.5$)		B=4 ($\mathcal{V}^* = NA$)		B=5 ($\mathcal{V}^* = NA$)	
	sol	time	sol	time	sol	time	sol	time	sol	time
BFS	1	50.51	1	582.92	0	NA	0	NA	0	NA
DC	1	41.40	1	**104.05**	0.4	181.56	0	NA	0	NA
BFD-rel-env	1	53.06	1	178.98	0.4	179.34	0	NA	0	NA
BFD-rel-mod	1	64.64	1	180.38	0.4	183.43	0	NA	0	NA
BFD-rel-comb	1	55.84	1	175.23	0.4	169.58	0	NA	0	NA
BFD-rel-proc	1	57.52	1	177.61	0.4	**167.40**	0	NA	0	NA
BFD-rel-comb-proc	1	52.05	1	178.52	0.4	179.48	0	NA	0	NA

Table 6: Exploding Blocks World (EX. BLOCK) (h_{MIN})

	B=1 ($\mathcal{V}^* = 4.1$)		B=2 ($\mathcal{V}^* = 3.9$)		B=3 ($\mathcal{V}^* = NA$)		B=4 ($\mathcal{V}^* = NA$)		B=5 ($\mathcal{V}^* = NA$)	
	sol	time	sol	time	sol	time	sol	time	sol	time
BFS	0.8	257.25	0	NA	0	NA	0	NA	0	NA
DC	0.8	**244.23**	0.3	645.34	0	NA	0	NA	0	NA
BFD-rel-env	0.7	354.32	0	NA	0	NA	0	NA	0	NA
BFD-rel-mod	0.8	365.9	0	NA	0	NA	0	NA	0	NA
BFD-rel-comb	0.7	362.63	0	NA	0	NA	0	NA	0	NA
BFD-rel-proc	0.8	279.67	0.3	**603.34**	0	NA	0	NA	0	NA
BFD-rel-comb-proc	0.8	304.12	0.3	625.90	0	NA	0	NA	0	NA

Table 7: Boxworld (BOX) (h_{BAOD})

design nodes.

We attribute this result to the benefit of using caching to avoid redundant computation of costly design nodes. Since the design nodes are closer to the root of the search tree, the calculation of their heuristic value is typically more costly than that of nodes that are closer to the leaf nodes. Accordingly, the saving in computation time achieved by avoiding their repeated computation is substantial.

We next analyze and compare the quality of the h_{MIN} and h_{BAOD} heuristics. For most instances, the less informative but easier to compute h_{BAOD} heuristic outperformed the h_{MIN} heuristic and was used by the dominating approach. The exception is for EX. BLOCKS, where we see that the added informative value of the h_{MIN} heuristic leads to an overall lower running time.

To understand this trend, we compared the number of nodes examined by each heuristic, to the number of unique nodes for which the heuristic value is calculated. To avoid repeated computations of previously encountered states, both of the heuristics used saved values of previously encountered nodes. The results show that in all domains, the methods that use h_{MIN} do not necessarily examine less nodes, but always evaluate the value of a substantially smaller number of unique nodes. For example, for the Ex. BLOCKS domain with the rel-proc heuristic, we see that for budget of 1 and heuristic h_{BAOD}, the average number of nodes examined is 188535.6, while the number of unique nodes is 29441.5. For h_{MIN}, the average number of examined nodes is 518671.5, out of which 2809.6 are unique. The fact that the additional computational effort required to achieve the added accuracy of the h_{MIN} heuristic was beneficial in one of the domains we examined, stresses the importance of finding the right balance between the computation time invested in evaluating each node, and the value of high accuracy heuristic estimations that can prevent the exploration of redundant nodes.

Finally, we note that some of our examined problems, like BOX with budget greater then 2, could not be solved by any of the methods. These results are due to the time limit we set on the computation of each instance and the limitation of our evaluation resources, but have no implication on the efficiency of the examined approaches.

Related Work

Environment design (Zhang, Chen, and Parkes 2009) provides a generic template for defining problems that involve modifying environments in order to maximize some utility. In addition, Zhang, Chen, and Parkes (2009) provide techniques for redesign for the specific special case in which the design objective is *policy teaching*, *i.e.*, influencing the agent to adopt a particular policy whose execution maximizes the utility of the designer as an interested party. Keren et al. (2017) formulated ER-UMD as a different and distinct special case of environment design where the objective is to find a sequence of modifications that maximize a shared utility of agent and designer.

For solving ER-UMD settings, two methods were suggested by Keren et al. (2017), namely the *DesignComp* compilation (DC) that embeds the design problem into a planning problem and a heuristic search (BFD) in the space of modifications. For the latter, they suggest applying modifications to a relaxed environment and show it generates an admissible heuristic.

We extend this approach by offering a set of heuristics based on the relaxation of the design process. By searching in the relaxed modification space, we potentially avoid the need to calculate the value of every possible modification, and use cached values to estimate the value of similar modifications. Our approach can be seen as complementary to the previous approaches, since caching and modification relaxation can be combined with environment relaxation to yield estimations that may be computed efficiently.

The relationship between the agent and designer utility dictates the types of methods that can be used to solve each environment design problem. In particular, ER-UMD exploits the correlation between agent and designer utilities to develop planning-based methods for design. The heuristics we propose (and show to be admissible) are not admissible for environment design problems in general and in particular not for the policy teaching setting addressed by Zhang and Parkes (2008), nor for many other variants of environment design such as goal recognition design (Keren, Gal, and Karpas 2014), where redesign is performed to enhance the ability to recognize the objective of an acting agent, with its own utility measure. Similarly, the design methods suggested by Zhang and Parkes (2008) and Keren, Gal, and Karpas (2014), do not apply to the ER-UMD setting we address.

Conclusions

This work proposed a new class of heuristics for ER-UMD, called *simplified-design*, which relax the modification pro-

cess by mapping each examined modification during the redesign process to a modification that *dominates* it. Instead of the original modification, we calculate the value of the dominating one, and cache the computed value for future use on nodes mapped to the same dominating modification. We identified conditions under which this heuristic class is admissible and discussed automatic generation of modification relaxations. Our empirical results showed the computational savings achieved by our suggested approaches.

For future work, we intend to automate the process of selecting the best relaxation approach for a given domain. In particular, this automated process will extract domain specific information to choose the best way to apply padding and the best underlying heuristic for solving each modified model. In addition, we intend to extend our *simplified-design* heuristic approach by adopting a hierarchical model and allow the search to alternate among different levels of *relaxation granularity*; for padded modification sequences that yields a utility gain, a more accurate (and costly) estimation is acquired while for padded sequences that leave the initial utility unchanged, we use the high level value. Finally, following the successful implementation of our approach on standard benchmarks, we seek additional settings and real-world applications.

References

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* 24:933–944.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI)*.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In *International Workshop on Model Checking and Artificial Intelligence*, 35–50. Springer.

Hansen, E. A., and Zilberstein, S. 1998. Heuristic search in cyclic AND/OR graphs. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 412–418.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; Koenig, S.; et al. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI)*.

Keren, S.; Pineda, L.; Gal, A.; Karpas, E.; and Zilberstein, S. 2017. Equi-reward utility maximizing design in stochastic environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

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

Pearl, J. 1984. Heuristics: intelligent search strategies for computer problem solving. *Addison-Wesley Pub. Co., Inc., Reading, MA*.

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)*.

Younes, H. L. S., 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. 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. 2009. A general approach to environment design with one agent. In *Proceedings of the Joint Conference on Artifical Intelligence (IJCAI)*.