Relaxed Modification Heuristics for Equi-Reward Utility Maximizing Design

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
sarahn@campus.technion.ac.il, lpineda@cs.umass.edu*

Abstract

Grounded in a stochastic setting, the objective of equi-reward utility maximizing design (ER-UMD) is to find a valid modification sequence, from a given set of possible environment modifications, which yields maximal agent utility. To efficiently traverse the typically large space of possible modification options, we use heuristic search and propose new heuristics, which relax the design process so 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 this approach is guaranteed to produce admissible estimates and describe a range of models that comply with these requirements. In addition, we provide simple methods to automatically generate dominating modifications. We evaluate our approach on a range of settings for which our heuristic is admissible and compare its efficiency with that of a previously suggested heuristic that employs a relaxation of the environment and a compilation from ER-UMD to planning.

Introduction

Equi-reward utility maximizing design (ER-UMD) (Keren *et al.* 2017) involves redesigning stochastic environments to maximize agent performance. The input of a ER-UMD problem consists of a description of a stochastic environment, a utility measure of the agents acting within it and the possible ways to modify the environment. The objective is to find a modification sequence that maximizes agent utility. The design process is viewed as a search in the often exponential space of possible modification sequences, which motivates the use of heuristic estimations to guide the search.

In this work we 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 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 sequences are cached. When a modification is mapped to a previously encountered relaxed modification, the cached value is reused. Of course, 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 and 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.

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 number of allowed modifications is constrained by a design budget.*

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 the 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*

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

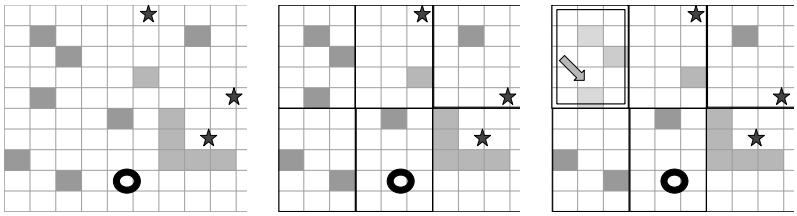

Figure 1: An example ER-UMD problem

same cell, the already computed value is reused.

The main contributions of this work are threefold. First, we propose a new class of heuristics for ER-UMD, called *simplified-design*. 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 demonstrates the benefit of the proposed heuristic.

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).

Background: Equi-Reward Utility Maximizing Design as Heuristic Search

The *equi-reward utility maximizing design* (ER-UMD) problem, recently suggested by 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 constraints. The aim is to find an optimal sequence of modifications to apply to the environment for maximizing the agent utility[1] under the given 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 $f : S \times A \times S \to [0, 1]$ 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 $\omega = \langle \epsilon^0, \mathcal{R}, \gamma, \mathbf{\Delta}, \mathcal{F}, \Phi \rangle$ where, $\epsilon^0 \in \mathcal{E}$ is an initial environment, $\mathcal{R} : \mathcal{S}_\mathcal{E} \times \mathcal{A}_\mathcal{E} \times \mathcal{S}_\mathcal{E} \to \mathbb{R}$ 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, and γ is a discount factor in $(0, 1]$, representing the deprecation of agent rewards over time. The set $\mathbf{\Delta}$ contains the 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. $\delta_i \in \mathbf{\Delta}$ and $\vec{\mathbf{\Delta}}$ is the set of all such sequences. $\mathcal{F} : \mathbf{\Delta} \times \mathcal{E} \to \mathcal{E}$ is a deterministic modification transition function, specifying the result of applying a modification to an environment. Finally, $\Phi : \vec{\mathbf{\Delta}} \times \mathcal{E} \to \{0, 1\}$ specifies allowed modification sequences in an environment.

[1]Whenever agent utility is expressed as cost, the objective is to minimize expected cost.

The reward function $\mathcal{R}$, discount factor γ and environment $\epsilon \in \mathcal{E}$ 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 and let $\mathcal{V}^*(\omega)$ represent the discounted expected agent reward of following an optimal policy from the initial state s_0 in an initial environment ϵ^0. The objective is to find a legal modification sequence $\vec{\delta} \in \vec{\mathbf{\Delta}}^*$ to apply to ϵ^0 that maximizes $\mathcal{V}^*(\omega^{\vec{\delta}})$ under the constraints, where $\omega^{\vec{\delta}}$ is the ER-UMD that results from applying $\vec{\delta}$ to ϵ^0.

Keren *et al.* (2017) propose to view the design process as a search in the space of modification sequences and suggest two methods for solving the ER-UMD problem. The first, referred to as *DesignComp*, embeds the offline design stage into the definition of the agent's planning problem which can be solved by any off-the-shelf solver. The second approach, namely the Best First Design (BFD) algorithm, applies a heuristic search in the space of modifications. To evaluate 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 (Holte *et al.* 1996)), 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.

After formally defining the *simplified-design* heuristic, we characterize ER-UMD settings where relaxing modifications 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.

We let ϵ^δ represent the environment that results from applying δ to ϵ, and let $\vec{\mathbf{\Delta}}$ and $\mathcal{E}$ represent all modification sequences and environments in a ER-UMD model, respectively.

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

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

where ω^δ *and* $\omega^{\delta'}$ *are the ER-UMD models that have* ϵ^δ *and* $\epsilon^{\delta'}$ *as their initial environments, respectively.*

The *simplified-design* heuristic, denoted by h^{simdes} estimates the value of applying a modification δ to ω by the value of the dominating modification δ'.

$$h^{simdes}(\omega^\delta) \stackrel{def}{=} \mathcal{V}^{max}(\omega^{\delta'}) \quad (1)$$

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*.

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 modification potentially reduces the computational overhead of the actual setting.

Another type of a dominating modification is created via *modification padding*, which involves appending to the explored modification a sequence of modifications.

Definition 2 (padded modification) *Given an ER-UMD model* $\omega = \langle \epsilon^0, \mathcal{R}, \gamma, \mathbf{\Delta}, \mathcal{F}, \Phi \rangle$*,* $\vec{\delta} = \langle \delta_1, \ldots, \delta_n \rangle$ *is a* padded modification *of* $\delta \in \mathbf{\Delta}$ *in* ω *if* $\exists i \ 1 \leq i \leq n$ *s.t.* $\delta = \delta_i$*.*

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 no impact on the agent's 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, by the value of removing all pieces of furniture from an entire cell (a black rectangular in Figure 1(right)).

Naturally, both 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*, which 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. 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-estimate a modification and can therefore be used to extract admissible estimates. Formally,

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

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

Definition 4 (independent model) *An ER-UMD model* ω *is* independent *if for any modification sequence* $\vec{\delta} \in \vec{\mathbf{\Delta}}$*, 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* $\vec{\delta}$*,*

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

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

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.

Automatic Dominating Modification Generation

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 that share the same values on a set of lifted parameters. In Example 1, the lifted representation of furniture removal modifications is represented by $ClearCell(x, y)$, where parameters x and y denote the cell coordinates. The value of the grounded modification $ClearCell(1, 3)$ can be (over)estimated by the value of applying the sequence $ClearCell(1, 1)$, $ClearCell(1, 2)$, $ClearCell(1, 3)$, *etc.* This value is cached, so when modification $ClearCell(1, 2)$ is examined, it is mapped to the same padded sequence, and the pre-computed value can be reused.

In models were the essence of modifications involve changing the probability distribution of an action's outcome, we can automatically create a relaxation by creating a separate action for each of the outcomes (known in the literature as *all outcome determinization*). 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.

Modifications for Independent Monotonic ER-UMD models

To characterize monotonic and independent models where modification padding can be used to produce admissible estimates, 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) *A modification δ is an* action addition *modification (ADM) if for any environment $\epsilon \in \mathcal{E}$, ϵ^δ 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. $app(s, \epsilon^\delta) = app(s, \epsilon) \cup A_{s,\delta}$.*

In Example 1, action addition is exemplified by enabling safe transitions between nearby states 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 modification used by 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. For example, 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.

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). We examined the benefits of using heuristic search and caching for utility maximizing design and analyze the role of different heuristics in solving the underlying MDPs.

We used a total of 20 instances from four PPDDL domains (5 of each), adapted from Keren *et al.* (2017) that included three stochastic shortest path MDPs with uniform action cost domains from the probabilistic tracks of the eighth International Planning Competition: 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 utility maximizing design setting and the ability to improve upon an initial design. In all domains, agent utility is expressed as expected cost and constraints as a design budget. For each domain, we used the modifications described by Keren *et al.* (2017), implementing all of them as action addition modifications (see Section for a detailed explanation).

Setup Evaluation was performed using optimal and suboptimal solvers within a time bound of five minutes. Each instance was solved using the following approaches:

- BFS - an exhaustive breadth first search in the space of modifications.
- *DesignComp* (Keren *et al.* 2017) (DC)- a compilation of the design problem to a planning problem, which embeds the design into the domain description.
- BFD (Keren *et al.* 2017) - Best First Design, a heuristic best first search in the space of modifications. For this approach we examined five heuristic approaches, the first of which was presented by Keren *et al.* (2017) and the other four are variations of the heuristic proposed in this work.
  - **rel-env** the *simplified-environment* heuristic where node evaluation is done on a relaxed environment.
  - **rel-mod** the *simplified-design* heuristic that estimates the value of a modification by a single dominating modification.

			BFS		DC		BFD rel-env		BFD rel-mod		BFD rel-combined		BFD rel-proc		BFD rel-combined-proc	
		V^*	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
BLOCKS	B=1	0.91	1.5	2709.4(95.2)	1.27	1624.2(559.4)	1.97	2709.4(95.2)	1.7	2709.4(95.2)	1.9	2709.4(95.2)	1.59	2611 (95.2,7.3)	1.7	2611 (95.2,7.3)
	B=2	0.91	2.58	42854.5(2483.5)	2.55	24442.3(7440.2)	3.43	42854.5(2483.5)	3.01	42854.5(2483.5)	3.5	42854.5(2483.5)	2.67	40396.4(2483.5,25.4)	2.86	40396(2483.5,25.4)
	B=3	0.91	37.6	441153.2(35901.4)	42.4	244523.5(67939.8)	59.48	441153.2(35901.4)	48.4	441153.2(35901.4)	58.46	441153.2(35901.4)	41.62	441153.2(35901.4,63.6)	35.8	441153.2(35901.4,63.6)
EX. BLOCKS	B=1	0.23	30.82	15724.4(41.4)	25.61	10839(118.6)	31.38	15724.4(41.4)	35.82	15724.4(41.4)	36.57	15724.4(41.4)	35.63	15720(41.4,7.6)	35.92	15720(41.4,7.6)
	B=2	0.01	272.2	13171.9(458.7)	45.55	2794.5(819.5)	275.9	13171.9(458.7)	275.3	13171.9(458.7)	282.3	13171.9(458.7)	171.7	8812.4(427.7,25.8)	185.6	8812.4(427.7,25.8)
	B=3	0.01	TO	TO	17.09	6251.2(3541.4)	TO	TO	TO	TO	TO	TO	527.832	1452884(2498,63)	523.2	1452884(2498,63)
TIRE	B=1	0.86	0.9	2343.2(35.5)	0.5	1074.2(79)	1.4	2343.2(35.5)	1.4	2343.2(35.5)	1.4	2343.2(35.5)	1.4	2343.2(35.5,5.4)	0.9	2313(35.5,5.4)
	B=2	0.83	0.58	14189.2(333.4)	0.38	6418.4(479.2)	0.67	14189.2(333.4)	0.61	14189.2(333.4)	0.62	14189.2(333.4)	0.6	14189.2(333.4,16.5)	0.6	14189.2(333.4,16.5)
	B=3	0.81	2.4	54343.41(1741.2)	1.6	54343.41(1741.2)	2.5	54343.41(1741.2)	2.4	54343.41(1741.2)	2.4	54343.41(1741.2)	2.3	54343.41(1741.2,34.3)	2.3	54343.41(1741.2,34.3)
VACUUM	B=1	0.75	6.25	5553.2(14.3)	0.91	977.4(15)	7.43	5553.2(14.3)	7.8	5553.2(14.3)	7.8	5553.2(14.3)	6.7	5542(14.3,3.4)	7.3	5542(14.3,3.4)
	B=2	0.67	18.24	15367.2(56.3)	4.68	3079.5(61.6)	23.68	15367.2(56.3)	19.04	15367.2(56.3)	31.93	15367.2(56.3)	18.59	15317.5(56.4,6)	25.5	15317.5(56.4,6)
	B=3	0.56	43.48	29257.4(150.4)	21.36	7140(182.2)	66.87	29257.4(150.4)	33.8	29257.4(150.4)	73.26	29257.4(150.4)	33.08	29115.4(150.4,8.2)	47.86	29115.4(150.4,8.2)

Table 1: Running time and expanded node count for optimal solvers with h_{BAOD}

			BFS		DC		BFD rel-env		BFD rel-mod		BFD rel-combined		BFD rel-proc		BFD rel-combined-proc	
		V^*	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
BLOCKS	B=1	0.91	1.49	2487.4(95.2)	1.97	1938.5(938.5)	1.43	2487.4(95.2)	1.9	2487.4(95.2)	4.4	2487.4(95.2)	1.9	2399.2(95.2,7.4)	4.4	2399.2(95.2,7.4)
	B=2	0.91	1.92	39425.4(2485.5)	1.89	27891.2(12867.4)	9.35	39425.4(2483.5)	3.4	39425(2483.5)	9.25	39425(2483.5)	4.2	36967(2483.5,25.4)	9.7	36967(2483.5,25.4)
	B=3	0.91	27.88	406762.4(35901.4)	38.93	271487.2(114492.2)	117.4	406762.5(35901.4)	59.60	406762.6(35901.4)	127.93	406762.5(35901.4)	54.72	370924.7(35901.4,63.6)	131.35	370924.7(35901.4,63.6)
EX. BLOCKS	B=1	0.23	44.23	505293.2(41.4)	34.83	124678.4(1354)	46.9	505293.2(41.4)	47.89	505293.2(41.4)	46.72	505293.2(41.4)	50.63	505259.2(41.4,7.3)	49.6	505259.2(41.4,7.3)
	B=2	0.01	344.32	3916380.5(458.4)	45.97	42741.6(9459.6)	348.8	3916380.5(458.4)	220.3	2551503.3(427.4)	231.4	2551503.3(427.4)	231.4	2551101.6(427.4,25.4)	225.5	2551101.6(427.4,25.4)
	B=3	0.01	TO	TO	43.2	59802.2(26763.4)	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO
TIRE	B=1	0.86	1.4	2920.6(35.5)	0.7	2920.6(35.5)	1.2	2920.6(35.2)	1.2	2920.6(35.5)	1.5	2890.6(35.5)	1.2	2890.6(35.5,5.4)	1.5	2890.6(35.5,5.4)
	B=2	0.83	0.63	17635.4(333.4)	0.4	17635.4(333.4)	0.84	17635.4(333.4)	0.72	17635.4(333.4)	0.96	17635.4(333.4)	0.7	17635.4 (333.4,16.5)	0.8	17635(333.4,16.5)
	B=3	0.81	2.5	67464.2(1741.2)	1.7	31450.2(4702.2)	3.03	67464.2(1741.2)	2.7	67464.2(1741.2)	3.4	67464.2(1741.2)	2.7	65760.4(1741.2,37.3)	3.4	65760.4(1741.2,37.3)
VACUUM	B=1	0.75	7.91	6903.4(14.3)	9.02	1200.2(227.2)	14.71	6903.4(14.3)	17.67	6903.4(14.3)	37.69	6903.4(14.3)	18.03	6892.2(14.3,3.4)	36.03	6892.2(14.3,3.4)
	B=2	0.67	37.4	18617.4(56.3)	52.3	4181(202)	99.23	18617.3(56.3)	91.93	18617.3(56.3)	394.29	18617.3(56.3)	91.44	18567.4(56.3,6.2)	411.76	18567.5(56.3,6.2)
	B=3	0.56	83.46	34181.4(150.4)	79.04	10165.2(442.2)	264.49	34181.4(150.4)	228.66	34181.4(150.4)	1034.48	34181.4(150.4)	204.98	34039.2(150.4,8.2)	225.6	34039.2(150.4,8.2)

Table 2: Running time and expanded node count for optimal solvers with h_{MinMin}

- **rel-combined** the *simplified-design* heuristic that estimates the modification value 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).
- **rel-combined-proc** the *simplified-design* heuristic that estimates the value of a modification using parametrized padding of relaxed modifications (on the first parameter of a modification).

Optimal solutions were acquired using a deterministic best first search for the design space for the BFD, and LAO* (Hansen and Zilberstein 1998) for the BFD execution nodes and DC (compilation) with convergence error bound of $\epsilon = 10^{-6}$. Approximate solutions were obtained by replacing LAO* with FLARES (Pineda *et al.* 2017), a sampling-based SSP solver that uses short-sighted labeling to cache the value of explored states and speed-up computation; we used the parameters $t = 0$, $\epsilon = 0.01$, and a maximum of 100 iterations. We also assigned a cap of 500.0 for the maximum state cost (100.0 for the approximate case), in order to handle dead-ends (Kolobov *et al.* 2012). In the approximate solution of the compilation approach, the expected cost is computed from 100 simulations of the policy computed by FLARES. For BFD, the cost after applying modifications is the state cost estimated by FLARES, which is guaranteed to be admissible.

Optimal solutions were tested on an Intel(R) Xeon(R) CPU X5690 machine with a budget of $1 - 3$. Approximate solutions were tested on Intel(R) Xeon(R) CPU E3-1220 3.40Ghz, with a budget of $1 - 3$. To implement the budget constraint we added a counter verifying the number of design action does not exceed the budget. Each run had a 30 minutes time limit.

For solving the underlaying MDP, we used two heuristic. The *Min-Min* heuristic (Bonet and Geffner 2005)(h_{MinMin}) solves the all outcome determinization using the zero heuristic. We also implemented the *bounded all-outcome determinization* (heuristic h_{BAOD}) performs a depth-bounded BFS exploration of the all outcome determinization. Both heuristics are admissible.

Results Separated by domain and budget, Table 1 and Table 2 summarize the average results acquired for each domain and each budget ($B = i$) using an optimal solver with h_{BAOD} and h_{MinMin} as the MDP heuristics, respectively. The tables present V^* as the reduction in expected utility for each design budget with respect to the initial utility (the values are the ratio with respect to the initial value). In addition, they present the running time in seconds ($time$) and the number of expanded nodes (evaluated by the heuristic) during the search ($nodes$). The number of design nodes, representing a modification sequence being applied, is in parenthesis. For the **rel-proc** and **rel-combined-proc** heuristics the numbers in parenthesis represent the number of expanded design node and the number of explicitly calculated design nodes, nodes for which the heuristic value of the dominating modification could not be found in the cache. TO indicates a time out for problems that exceeded the time bound.

Table 3 specifies the results acquired using the approximate solver with h_{BAOD} as the MDP heuristic. V^* represents the ratio between the simulated value and the one acquired using the optimal solver, $stderr$ represents the standard deviation, while $time$ and $nodes$ have the same meaning as in the optimal solver's tables.

Our first observation is that with regards to the optimal solutions, the compilation (DC) approach outperforms the BFD approach for most domains with a shorter running

		BFS				DC				BFD rel-env				BFD rel-mod				BFD rel-combined				BFD rel-proc				BFD rel-combined-proc			
		V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes	V^*	stderr	time	nodes
BLOCKS	B=1	1.05	0.45	0.28	4316.2(190.3)	0.95	0.43	0.48	3827.3(652.6)	1.09	0.48	0.3	4242.5(190.3)	1.14	0.49	0.28	4229.3(190.3)	1.14	0.49	0.3	4152.4(190.3)	1.07	0.45	0.28	4036.3(190.3,7.4)	1.07	0.46	0.288	4036.4(190.3,7.4)
	B=2	1.07	0.49	6.27	65772.2(4966.2)	1.02	0.44	4.02	58102.5(9388)	1.06	0.46	6.24	65772(4966.2)	1.02	0.43	5.48	66093(4966.2)	1.02	0.47	5.58	66089(4966.2)	1.02	0.45	6.5	60623(4966.2,25.5)	1.02	0.43	6.47	60623(4966.2,25.5)
	B=3	1.02	0.43	80.68	693387.5(71802.2)	1.02	12.68	58.4	582887.5(96126.9)	1.03	0.46	101.24	692854.5(71802.2)	1.02	0.45	93.78	692558.6(71802.2)	1.07	0.45	105.49	693245.6(71802.2)	1.01	0.41	82.42	622343.2(71802.2,63.5)	1.2	0.49	82.73	621769.5(71802.2,63.5)
EX. BLOCKS	B=1	1.9	23.98	12.47	134918.4(82.2)	1.17	16.49	2.8	5601.2(170.2)	1.9	12.3	12.58	134191(82.2)	1.2	16.08	42.77	132526.4(82.2)	1.9	21.2	11.7	130487.7(82.2)	1.9	13.06	13.2	129957.5 (69.4,7.5)	1.9	23.47	11.55	136625.5(69.4,7.5)
	B=2	0.01	0	73.5	768204.4(916.3)	0.01	0	16.31	25329.6(1226.3)	0.01	0	71.71	771082.4(916.3)	0.01	0	65.07	778786.5(916.3)	0.01	0	76.8	782888.7(916.3)	0.01	0	31.5	180761(420.7,25.9)	0.01	0	50.4	495917.8(455.7,25.9)
	B=3	0.01	0	443.3	2935654.4(5622.3)	0.01	0	114.9	98556.5(6009.4)	0.01	0	426.4	2940276.5(5622.2)	0.01	0	385.6	2909483.4(5622.3)	0.01	0	288.58	2925312.2(5622.3)	0.01	0	100.4	338548(2526,63)	0.01	0	102.6	334572(2526,63)
TIRE	B=1	1.48	0.2	0.41	2612.3(70.2)	2.1	24.779	0.03	1083(171.6)	1.09	0.49	0.43	2982(70.2)	1.02	0.49	0.42	3351.5(70.2)	1.02	0.48	0.43	3593.5(70.2)	1.18	0.2	0.41	2998(635.4,5.6)	1.17	0.49	0.42	3329.5(63.4,5.5)
	B=2	1.03	0.45	0.7	16672.4(666.4)	2.1	24.9	0.24	5087.5(916.5)	1.03	0.47	0.65	16608.8(666.4)	1.02	0.44	0.67	17618.6(666.4)	1.03	0.48	1.2	16580.8(666.4)	1.05	0.49	0.68	17547.5(614.6,16.4)	1.04	0.43	0.64	15789.7(494.3,16.4)
	B=3	1.02	0.45	2.52	58570.4(3482.4)	2.4	22.4	0.7	19412.5(3535.3)	1.03	0.45	2.52	58570.5(3482.2)	1.15	0.45	2.37	56487.6(3482.4)	1.13	0.21	2.54	58807.8(3482.9)	1.03	0.49	2.42	57306.7(3402.5,37.3)	1.04	0.48	2.42	55508(3407.4,37.3)
VACUUM	B=1	1.45	0.47	11.21	6075.6(728.3)	1.06	0.45	2.75	2090.3(27.5)	1.51	0.48	10.46	5848.6(28.5)	1.05	0.43	9.08	5140.3(28.5)	1.04	0.44	10.2	4917.6(28.5)	1.03	0.43	8.32	4873.6(28.5,3.3)	1.04	0.43	9.4	4895(28.5,4.4)
	B=2	1.44	0.47	26.01	13414.5(1121.4)	0.01	0.43	9.08	6736.5(102.6)	1.01	0.41	28.7	11937.5(112.4)	1.01	0.42	22.74	11679.6(112.4)	0.01	0.42	37.52	11981.7(112.4)	1.4	0.48	25.67	13028.5(112.4)	1.4	0.48	25.7	13028.5(112.4)
	B=3	1.4	0.49	33.84	20071.5(300.4)	1.01	0.43	17.95	13767.6(233.2)	1.46	0.49	16.9	19752.6(300.4)	1.4	0.46	51.92	20306.7(300.4)	1.4	0.46	109.841	20306.7(300.4)	1.01	0.41	46.4	18364.7(300.4)	1.02	0.45	70.3	18372.7(300.4)

Table 3: Running time for sub-optimal solvers using the h_{BAOD} heuristic

time and less expanded nodes. The only exception occurs in the BLOCKS domain for budget 3 and the h_{BAOD} MDP heuristic, for which **rel-combined-proc** outperforms the other approaches and the same setting with h_{MinMin} as the MDP heuristic, for which the BFS approach was best. It is worth noting, however, that the number of design nodes, each corresponding to a modification sequence, is higher for DC then for all BFD approaches.

Our evaluation uses only independent models. Therefore, for any budget above 1, the BFD approach examines all the permutations of a given modification sequence separately, while for the compilation, the value for these nodes is computed only once. However, the use of independent models does not explain the superior performance of the compilation over the BFD approach for budget 1 as well.

To examine this trend further, we compared the number of nodes that are evaluated by the heuristic to the distinct nodes evaluated for the first time (and for which the heuristic value has not been computed). The results show that while the DC examines up to 20% less nodes, the number of distinct nodes for both BFD and DC is similar. We also performed additional evaluations on small instances of the VACCUM domain (2×2 and 3×3) where the BFD approaches, and the **rel-proc** procedure in particular, outperformed the compilation in terms of both running time and expanded nodes. These results show that the efficiency of the applied approach depends on the specific problem structure and indicate that further investigation of both the nature of the benchmarks and the LAO* algorithm are warranted to understand the results and evaluate the efficiency of our proposed methods.

Next, we analyze the use of caching by comparing **rel-env**, **rel-mod**, and **rel-combined** that do not use caching against their counterparts **rel-proc** and **rel-combined-proc** that are applied to a relaxed environment and re-use previously computed costs. The results show the newly proposed heuristics outperform the heuristics proposed by Keren *et al.* (2017) for all instances in terms of running time. This is due to saving in computation gained by the caching of similar modifications. This applies to both h_{MinMin} and h_{BAOD} heuristics.

Comparing h_{BAOD} (Table 1) with h_{MinMin} (Table 2) we note that for most instances the h_{BAOD} outperformed the h_{MinMin} heuristic, both in terms of running time and the number of explored nodes.

Exploring the different heuristic approaches for BFD calculation, we observe that for most domains, the different approaches yield the same number of explored nodes as the blind search (indicated by BFS). The only approaches that reduce the number of calculated nodes are the caching-based approaches, namely **rel-proc** and **rel-combined-proc** that reduce the computational overhead by avoiding redundant computations. This suggests that the relaxations we apply are non-informative in the domains we explore, leaving us with the wish to explore other, more elaborate domains in which the value of the heuristic approaches will be demonstrated.

For the approximate solvers (Table 3), the results indicate that in most cases the solvers we have used managed to achieve a utility reduction that deviated from the optimal design by up to 10%. Notable in particular is the ability to achieve a nearly optimal design for the EX.BLOCKS domain, which could not be solved by all but the DC in the optimal setting. Indeed, as in the optimal case, the DC compilation is the dominating approach for most domains. However, results are inconclusive since in most cases they fail to provide a single computation method that outperforms the other approaches on all measures. This, again, indicates that further investigation is needed into the pros and cons of using sub-optimal solvers.

Related Work

Environment design (Zhang *et al.* 2009) provides a general framework for modifying environments with the objective of maximizing some utility. Keren *et al.* (2017) formulated ER-UMD as a special case of environment design where the objective is to find a sequence of modifications that maximize some agent utility.

For solving ER-UMD settings, two methods were suggested by Keren *et al.* (2017), namely a compilation (DC) that embeds the design problem into a planning problem and 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 modification padding technique we suggest to

generate 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.

As noted by Keren *et al.* (2017), it is the relationship between agent and system utility that dictates the types of methods that can be used to solve an environment design problem. In particular, for ER-UMD we exploit the correlation between agent and system utilities to develop planning-based methods for design. The heuristics we propose (and show to be admissible) are not admissible for environment design in general and in particular not for goal recognition design (Keren *et al.* 2014) or policy teaching (Zhang and Parkes 2008).

Conclusions

This work proposed a new class of heuristics for ER-UMD, called *simplified-design*, which relax the modification process by mapping each modification that is expanded during the redesign 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. We identified conditions under which this heuristic class is admissible and discussed automatic generation of relaxations.

For future work, we intend to automate the process of selecting the best relaxation approach for a given domain. In addition, we intend to implement an approach that may alternate during the search between different levels of *relaxation granularity*; for padded modification sequences that yields a utility gain, a more accurate (and costly) estimates is acquired while for padded sequences that leave the initial utility unchanged, we use the high level value.

References

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

Blai Bonet and Héctor Geffner. mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24:933–944, 2005.

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

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

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

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, Sven Koenig, et al. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, volume 7, pages 1007–1012, 2007.

Robert C. Holte, M. B. Perez, R. M. Zimmer, and Alan J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, pages 530–535, 1996.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 154–162, 2014.

Sarah Keren, Luis Pineda, Avigdor Gal, Erez Karpas, and Shlomo Zilberstein. Equi-reward utility maximizing design in stochastic environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2017)*, August 2017.

Andrey Kolobov, Daniel Weld, et al. A theory of goal-oriented mdps with dead ends. *arXiv preprint arXiv:1210.4875*, 2012.

Luis Enrique Pineda, Kyle Hollins Wray, and Shlomo Zilberstein. Fast ssp solvers using short-sighted labeling. In *AAAI*, pages 3629–3635, 2017.

Haoqi Zhang and David Parkes. Value-based policy teaching with active indirect elicitation. In *Proceedings of the Twenty-Third Conference on Artificial Intelligence (AAAI)*, pages 208–214, 2008.

Haoqi Zhang, Yiling Chen, and David Parkes. A general approach to environment design with one agent. In *Proceedings of the Twenty-First International Joint Conference on Artifical Intelligence*, pages 2002–2008, 2009.