Redesigning Stochastic Environments for Maximized Utility

Sarah Keren[1], Luis Pineda[2], Avigdor Gal[1], Erez Karpas[1] and Shlomo Zilberstein[2]

[1]Technion – Israel Institute of Technology
{sarahn@,avigal@ie,karpase@}.technion.ac.il,
[2]University of Massachusetts Amherst
{lpineda@,shlomo@}.cs.umass.edu

Abstract

We present the Utility Maximizing Design (UMD) model for optimally redesigning stochastic environments to achieve maximized performance. This model suits well contemporary applications that involve the design of environments where robots and humans co-exist an co-operate, *e.g.*, vacuum cleaning robot. We discuss two special cases of the UMD model. The first is the equi-reward UMD (ER-UMD) in which the agents and the system share a utility function, such as for the vacuum cleaning robot. The second is the goal recognition design (*GRD*) setting, discussed in the literature, in which system and agent utilities are independent. To find the set of optimal modifications to apply to a UMD model, we present a generic method, based on heuristic search. After specifying the conditions for optimality in the general case, we present an admissible heuristic for the ER-UMD case. We also present a novel compilation that embeds the redesign process into a planning problem, allowing use of any off-the-shelf solver to find the best way to modify an environment when a design budget is specified. Our evaluation shows the feasibility of the approach using standard benchmarks from the probabilistic planning competition.

Introduction

We are surrounded by environments that are designed and manipulated with the intention of maximizing some benefit. Hospitals may be designed to minimize the daily distance covered by staff, supermarkets are constantly rearranged to make sure users buy as much as possible, airports may be designed to increase passenger spending, computer networks are structured to maximize message throughput, *etc.*

Common to all these environments is that their design is controllable. Such environments can be designed and often later redesigned to accommodate a specific objective. In addition, such environments need to account for different forms of uncertainty.

We aim at providing a generic model to support the offline design of such environments. Therefore, we present a model of *Utility Maximizing Design* in which a problem of redesigning non-deterministic environments in order to maximize system utility is specified. Non-determinism is expressed by stochastic outcomes of actions performed by agents. The setting we propose takes as input a stochastic environment, a set of allowed modifications, a set of constraints and a system utility criteria. It then finds an optimal set of modifications to apply to the environment for maximizing expected utility under the constraints.

Example 1 *Consider Figure 1(left), where a vacuum cleaning robot is placed in a living room. The utility of the robot may be expressed in various ways; it may try to clean an entire room as quickly as possible or cover as much space as possible before its battery runs out. In any case, (re)moving a piece of furniture from or within the room (Figure 1(center)) may increase the robot's utility. Accounting for uncertainty, there may be specific locations in which the robot tends to slip, ending up in a different location than intended. Increasing friction,* e.g.*, by introducing a high friction tile (Figure 1(right)), may reduce the probability of undesired outcomes in particular locations. Both types of modifications are applied offline (since such robots typically perform their task unsupervised) and should be applied economically in order to maintain usability of the environment.*

The proposed *Utility Maximizing Design* (UMD) model is a general model whose instantiations provide common grounds for comparative analysis and identification of efficient methods for special cases. A key observation with the UMD model is that utility may differ between the system and the agents acting in it. While Example 1 illustrates an *Equi-Reward* UMD(ER-UMD) case where agent and system share a utility function, earlier works on goal recognition design (Keren *et al.* 2014; Wayllace *et al.* 2016)(*GRD*) assumed optimal agents while the system aims at minimizing expected goal recognition time. We show that different assumptions on the relation between agent and system utility induce different solution techniques.

In this work we assume a fully observable stochastic setting and use Markov decision processes (Bellman 1957) to model the agent environment. We offer a general solution to the redesign problem by using heuristic search that yields optimal design strategies when using admissible heuristics. We formulate the conditions for admissibility for UMD settings and propose a heuristic based on simplifications of the environment, which we show to be admissible for the ER-UMD case but not for *GRD*. In addition, for ER-UMD, we exploit the alignment of system and agent utility to show a

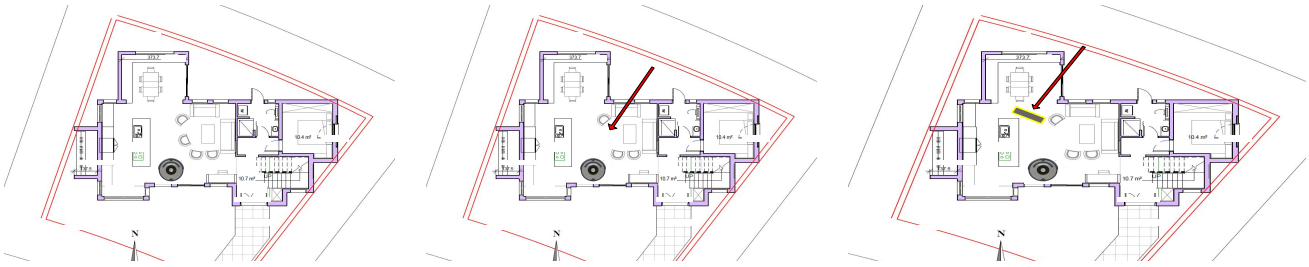

Figure 1: An example of a *Utility Maximizing Design* problem

way to piggyback on the search for optimal policy to find an optimal set of modifications.

We evaluate our work using probabilistic benchmarks from the International Planning Competitions, where a variety of stochastic shortest path MDPs are introduced (Bertsekas 1995). We show how redesign substantially improves utility, expressed via reduced cost achieved with a small modification budget. Moreover, the techniques we develop outperform the exhaustive approach.

The contributions of this work are threefold. First, we describe a new general model, namely *Utility Maximizing Design*, which involves the offline redesign stochastic environments for improving utility and show how goal recognition design is a special case of this setting. In particular, changing probability distributions offers a wide range of subtle (more realistic) modifications to be applied to a model, *e.g.*, reducing the probability of a slipping rather than eliminating it altogether. Second, we present a general method for solving UMD problems using informed heuristic search and specify the conditions under which an optimal solution can be found. Finally, for the special case where agent and system utility function is the same we formulate and compare three approaches for finding an optimal set of modifications to apply given a budget, namely an informed search approach with an admissible heuristic, a compilation-based method that embeds design into the definition of a planning problem.

Background

Non-deterministic planning problems with full feedback are typically modeled using a Markov decision process (MDP) (Bertsekas 1995). An MDP can be described in various ways, depending on the way agent utility and optimization criteria are defined (Mausam 2012; Kaelbling *et al.* 1998). Typically, an MDP is described by a tuple $\langle S, A, f, \mathcal{R}, \gamma \rangle$ where S is a finite set of states, $A(s) \subset A$ is the set of actions an agent can apply in state $s \in S$, $f : S \times A \times S \rightarrow [0, 1]$ is a transition function specifying the probability $f(s, a, s')$ of reaching state s' after applying action a in $s \in S$ and $\mathcal{R} : S \times A \times S \rightarrow \mathbb{R}$ is a function specifying the reward $\mathcal{R}(s, a, s')$ obtained when the system goes from state s to state s' as a result of executing action a (we assume a stationary reward function). The solution of an MDP is a control policy $\pi : S \rightarrow A$ describing the appropriate action to perform at each state in order to maximize the accumulated reward. Given a policy π and a state $s \in S$, the expected reward of following policy π from state s is described by $V^{\pi}(s)$. The optimal value of s is the solution of the Bellman equations (Bellman 1957): $V^*(s) = \max_{a \in A}[[\sum_{s' \in S} f(s, a, s')[\mathcal{R}(s, a, s') + \gamma V^*(s')]$.

To guarantee the expected reward is finite, additional restrictions are imposed. The *infinite horizon discounted reward* MDP is an MDP, as described above, with the restriction $0 < \gamma < 1$. Another restricted formulation is the *stochastic shortest path* MDP (SSP-MDP) (Bertsekas 1995) defined by the tuple $\langle S, A, f, \mathcal{C}, \mathcal{G}, s_0 \rangle$ where S, A and f are as above, $\mathcal{C} : S \times A \times S \rightarrow (0, \infty)$ specifies the strictly positive cost $\mathcal{C}(s, a, s')$ of applying action a in state s and ending up in state s', with the exception of goal states $\mathcal{G} \subseteq S$, which are all absorbing. Additionally, we have an initial state s_0. The objective is to find a policy that minimizes the expected cost of reaching a goal state from s_0.

Model

Having reviewed the necessary background, we now present the *Utility Maximizing Design* (UMD) model (Definition 4) that has three components: a stochastic environment in which agents act (ϵ), possible behaviors of agents in the environment and the ways by which they accumulate rewards (α), and the utility maximizing system that has control over the environment via applicable modifications (σ). We formulate each of these components separately before integrating them into the UMD model definition.

Definition 1 *An environment $\epsilon = \langle S_\epsilon, A_\epsilon, f_\epsilon \rangle$ is a triple with a set of states S_ϵ, a set of actions A_ϵ and a stochastic transition function $f_\epsilon : S_\epsilon \times A_\epsilon \times S_\epsilon \rightarrow [0, 1]$.*

Whenever ϵ is clear from the context we use S, A and f. We use $\mathcal{E}$ to denote a set of environments of interest. For example, we may wish to be able to move furniture around a room but not to break down walls. Given an environment $\epsilon \in \mathcal{E}$, we let Π_ϵ represent the set of all possible policies in ϵ. The feasibility of a policy may be dictated, for example, by the laws of physics. *e.g.*, a vacuum cleaning robot cannot move through an obstacle. In addition, $\Pi_\mathcal{E} = \bigcup_{\epsilon \in \mathcal{E}} \Pi_\epsilon$ represents the union of policies in all environments in $\mathcal{E}$.

In a given environment, agents may choose different behaviors depending on the utility measure they aim at maximizing. We characterize an agent using a reward function, describing the immediate reward collected by the agent at each possible transition, and a discount factor, dictating the way by which rewards are accumulated over time.

Definition 2 $\alpha = \langle \mathcal{T}, \{\mathcal{R}_\tau\}_{\tau\in\mathcal{T}}, \{\gamma_\tau\}_{\tau\in\mathcal{T}}\rangle$ *is a triple, defining a set of agent types* $\mathcal{T}$*. Given an environment* ϵ *and a type* $\tau \in \mathcal{T}$*:*

- $\mathcal{R}_\tau : S_\epsilon \times A_\epsilon \times S_\epsilon \to \mathbb{R}$ *is a Markovian and stationary reward function, specifying the reward* $r_\tau(s, a, s')$ *an agent of type* τ *gains from transitioning from state* s *to* s' *by the execution of* a *(a reward becomes a cost when negative).*
- γ_τ *is an agent-type specific discount factor in* $(0, 1]$*, representing the deprecation of agent rewards over time.*

Combining an environment ϵ, a reward function $\mathcal{R}_\tau(\epsilon)$, and a discount factor $\gamma_\tau(\epsilon)$ yields an MDP $\langle S_\epsilon, A_\epsilon, f_\epsilon, \mathcal{R}_\tau, \gamma_\tau\rangle$, leading to a set of MDPs that share the environment but differ in the way agent utility is defined. Whenever ϵ and τ are clear from context we use $\mathcal{R}$, and γ.

Given an environment $\epsilon \in \mathcal{E}$, we let $\Pi_{\langle\epsilon,\tau\rangle} \subseteq \Pi_\epsilon$ represent the set of policies agents of type τ may follow in ϵ. Accordingly, $\Pi_{\langle\epsilon,\mathcal{T}\rangle} = \bigcup_{\tau\in\mathcal{T}} \Pi_{\langle\epsilon,\tau\rangle}$ represents the union of policy sets of all agent types, thus representing the type of behaviors that may occur in a given environment.

We now turn our attention to the system point of view of an environment and the agents that act within it. The model supports a system that controls an environment by applying to it a sequence of modifications (atomic changes such as additions and deletion to environment elements) in order to maximize the system's utility. Accordingly, we let $\mathcal{M}$ represent a set of modifications and define a modification sequence $\vec{m} = \langle m_1, \ldots, m_n\rangle$ such that for all $1 \leq i \leq n, m_i \in \mathcal{M}$ is an ordered set of modifications. We let $\vec{\mathcal{M}}$ represent the set of all sequences of modifications over $\mathcal{M}$. We refer to sequences rather then sets to support settings where the order of application effects the impact on the model.

The system component σ specifies both the way the system accumulates rewards and the ways by which it can redesign the environment.

Definition 3 *A system* $\sigma = \langle \mathcal{R}_\sigma, \mathcal{M}_\sigma, \Theta_\sigma, \Phi_\sigma\rangle$ *is a quadruple, such that given an environment* ϵ *and an agent component* α*:*

- $\mathcal{R}_\sigma$ *is the system reward* $\mathcal{R}_\sigma : \Pi_{\langle\epsilon,\mathcal{T}\rangle} \to \mathbb{R}$*, which specifies the expected reward a system accumulates when an agent executes a policy* $\pi \in \Pi_{\langle\epsilon,\mathcal{T}\rangle}$*.*
- $\mathcal{M}_\sigma$ *is a finite set of modifications a system can apply. Modifications are associated with a system cost* $\mathcal{C}_\sigma : \mathcal{M}_\sigma \to \mathbb{R}^+$ *and a cost of a modification sequence is defined to be* $\mathcal{C}_\sigma(\vec{\mathrm{m}}) = \sum_{\mathrm{m}\in\vec{\mathrm{m}}} \mathcal{C}_\sigma(\mathrm{m})$*.*
- $\Theta_\sigma : \mathcal{M} \times \mathcal{E} \to \mathcal{E}$ *is a deterministic modification transition function, specifying the result of applying a modification on an environment.*
- $\Phi_\sigma : \vec{\mathcal{M}} \times \mathcal{E} \to \{0, 1\}$ *is an indicator that constrains the allowed sequence of modifications on an environment.*

Finally, the UMD model is defined as follows.

Definition 4 *A* Utility Maximizing Design *(UMD) model is a quadruple* $\langle \mathcal{E}, \alpha, \sigma, \epsilon^0\rangle$ *where* $\mathcal{E}$*,* α*,* σ *are as defined above and* ϵ^0 *is the initial environment.*

To specify the UMD redesign problem, we first define the system utility $\mathcal{U}_\sigma(\epsilon)$ to be the minimal expected system reward over all possible agent policies.

$$\mathcal{U}_\sigma(\epsilon) = \min_{\pi\in\Pi_{\langle\epsilon,\mathcal{T}\rangle}} \mathcal{R}_\sigma(\pi) \tag{1}$$

Given an environment ϵ and a sequence $\vec{m}$ of modifications such that $\Phi_\sigma(\vec{m}) = 1$, *i.e.*, $\vec{m}$ can be applied to ϵ, we let $\epsilon_{\vec{m}}$ represent the environment that is the result of applying $\vec{m}$ to ϵ. The general problem we aim to tackle is therefore

$$\underset{\vec{m}\in\vec{\mathcal{M}}_\sigma|\Phi_\sigma(\vec{m})=1}{\text{maximize}} \quad \mathcal{U}_\sigma(\epsilon^0_{\vec{m}}) \tag{2}$$

In particular, we are interested in a solution to Problem 2 which minimizes modification cost $C(\vec{m})$ of applying modification sequence $\vec{m}$. For a given UMD model δ, we denote a solution to Problem 2 by $\vec{\mathcal{M}}^*_\delta = \{\vec{m}^*\}$ and the maximal system utility by $\mathcal{U}^{max}(\delta)$.

It is worth noting that system reward is defined as an arbitrary function of the agent's policy which is given as an input parameter without any assumptions on the complexity of computing it. Moreover, while agent utility and system utility may be aligned, it is not necessarily the case. In Example 1, both the vacuum cleaning robot and the system design process share a common objective of maximizing expected agent reward. Conversely, in a goal recognition design setting, while agents either minimize the cost to their goals (Keren *et al.* 2014; b; Son *et al.* 2016) or aim to conceal them (Keren *et al.* a; 2016), system utility is defined by the maximal expected distance an agent can advance in the system before its goal is recognized. Next we describe these two settings as special cases of the UMD framework.

Equi-reward UMD

The *equi-reward utility maximizing design* (ER-UMD) model accounts for settings, like the one in Example 1, where a single reward maximizing agent type is defined and shares the same utility function with the system. The system chooses among the set of allowed modifications to redesign the environment in order to maximize user utility.

The ER-UMD is a special case of the UMD model defined in Definition 4 with an environment $\epsilon = \langle S_\epsilon, A_\epsilon, f_\epsilon, s_{0,\epsilon}\rangle$ that includes an initial state $s_{0,\epsilon} \in S_\epsilon$. The agents description α includes a single agent type $\mathcal{T} = \{\tau\}$ (hereon omitted from notation) with $\mathcal{R}$ and γ, the reward function and discount factor associated with agents, respectively.

We assume agents are optimal. Therefore, ϵ and α define a set of optimal policies $\Pi_{\langle\epsilon\rangle} = \Pi^*_{\langle\epsilon\rangle}$ an agent may follow. The system component σ describes the design process to maximize agent utility. Thus, given an environment ϵ and agent policy $\pi \in \Pi^*_{\langle\epsilon\rangle}$, system reward is defined as $\mathcal{R}_\sigma(\pi) = V^\pi_\epsilon(s_{0,\epsilon})$, where $V^\pi_\epsilon(s_{0,\epsilon})$ is the (optimal) expected reward from following π in ϵ starting from $s_{0,\epsilon}$.

Modifications $m \in M$ can be defined arbitrarily supporting all the changes applicable to a deterministic environment (Herzig *et al.* 2014). For example, we can allow adding the a transition between previously disconnected states. Particular to a stochastic environment is the option of modifying the transition function by increasing and decreasing the probability of specific outcomes. In Example 1, this corresponds to adding friction to slippery areas, thus increasing the probability of the agent ending up at its intended state.

The constraint set Φ_σ includes a single constraint: ϕ_B that specifies the modification budget B limiting the number of allowed modifications.

Since all possible agent policies share the same system utility function, the objective is to find a modification sequence $\vec{m}^* \in \vec{\mathcal{M}}^*$ which maximizes system utility $\mathcal{U}_\sigma(\epsilon^0_{\vec{m}^*}) = V^*_{\epsilon^0_{\vec{m}^*}}(s_{0,\epsilon^0_{\vec{m}^*}})$ under the budget constraint. In particular, we seek solutions which minimize modification cost.

Goal recognition design as UMD

Goal recognition design (*GRD*) aims at minimizing non-distinctive behavior, behavior that does not reveal the goal of the executing agent (Keren *et al.* 2014). We show that *GRD* is a special case of the UMD model in which a system, panelized for non-distinctive behavior, is redesigned by disallowing actions from the model.

Following (Wayllace *et al.* 2016), a stochastic *GRD* problem is represented using a stochastic shortest path MDP(SSP-MDP) with a set of possible goals states and a design budget that can be used to disallow actions in the model. A *GRD* setting is then defined by the tuple $R = \langle S_R, s_{0,R}, A_R, f_R, \mathcal{G}_R, \{C_{R,g}\}_{g\in\mathcal{G}_R}, B_R\rangle$, where S_R is a set of states, $s_{0,R}$ is the initial state, A_R are the set of actions, f_R is a stochastic transition function $f_R : S_R \times A_R \times S_R \to [0, 1]$, $\mathcal{G}_R \subseteq S_R$ is the set of possible goal states of an agent in R, $C_{R,g} : S_R \times A_R \times S_R \to \mathbb{R}^{0+}$ is a cost function specifying the cost of executing actions a in state s and arriving at state s' for agents aiming at goal $g \in \mathcal{G}_R$ and B_R is the maximal number of actions that can be disallowed in the model. A *GRD* setting is modified by disallowing a set of actions $\hat{A} \subseteq A_R$ under the constraints the expected cost to all goals is unchanged and $|\hat{A}| \leq B_R$. We use S, s_0, A, f, $\mathcal{G}$, $\{C\}_{g\in\mathcal{G}}$ and B when clear from context.

A goal recognition design setting is a special case of a UMD model, with an environment $\epsilon = \langle S_\epsilon, A_\epsilon, f_\epsilon, s_{0,\epsilon}\rangle$ for which the initial state $s_{0,\epsilon}$ is specified. The description of the agents component α specifies the set of types as possible goals $\mathcal{T} = \mathcal{G}$ and for every goal $g \in \mathcal{G}$, the reward $\mathcal{R}_g(\epsilon)$ is represented by the action cost function of the goal $C_g(\epsilon)$ and $\gamma_g(\epsilon) = 1$. Assuming optimality, the set $\Pi_{\langle\epsilon,g\rangle}$ for each agent type g is the set $\Pi^*_{\langle\epsilon,g\rangle}$ of optimal policies agents aiming at g may follow. Non-distinctive policies $\Pi^{nd}_{\langle\epsilon,g\rangle} \subseteq \Pi^*_{\langle\epsilon,g\rangle}$ are the prefixes of these policies that are shared by more than one goal (see (Wayllace *et al.* 2016) for a full description).

The system component $\sigma = \langle\mathcal{R}_\sigma, \mathcal{M}_\sigma, \Theta_\sigma, \Phi_\sigma\rangle$ describes the design process that disallows actions in order to minimize non-distinctive behavior, corresponding to executing non-distinctive policies. Each modification $m_a \in \mathcal{M}_\sigma$ corresponds to the removal of a single action $a \in A_R$ from the model. The modification function is then $\Theta_\sigma(m_a, \epsilon) = \epsilon'$ where $\epsilon' = \langle S_\epsilon, A_\epsilon \setminus a, f_\epsilon, s_{0,\epsilon}\rangle$.
The constraint set Φ_σ includes two constraints: ϕ_B specifying the modification budget B and ϕ_* disallowing a change to the expected cost to any of the goals from the initial state.

System reward $\mathcal{R}_\sigma(\pi)$ for policy π is defined as the expected agent cost of following the maximal non-distinctive prefix of π. Accordingly, system utility $\mathcal{U}_\sigma(\epsilon)$ is the *worst case distinctiveness*(*wcd*), the maximal expected cost over all non distinctive policies $\bigcup_{g\in\mathcal{G}} \Pi^{nd}_{\langle\epsilon,g\rangle}$.

The objective is to minimize *wcd* value by using a modification sequence $\vec{m} = \langle m_{a_1}, \ldots, m_{a_n}\rangle$ of actions to be disallowed in ϵ^0 that complies with the connstraints. In particular, we may be interested in the minimal set that achieves this.

Finding $\vec{m}^*$ using informed search

The baseline method for finding an optimal modification sequence to apply to a UMD model is an exhaustive exploration of all allowed modification sequences and selecting one that maximizes system utility. This approach was used for finding the optimal set of disallowed actions in a goal recognition design setting (Keren *et al.* 2014; Wayllace *et al.* 2016). There, pruning was applied as a way to reduce the size of the state space. The pruning of the *GRD* search tree is performed by exploring only modifications that disallow actions that are part of the set of *wcd* policies, policies that share the maximal non-distinctive prefix.

In this work, we propose a different approach. We note that the approach described above is not applicable to the general UMD model that supports arbitrary modification options that may affect system utility when applied to **any** part of the model. For example, removing furniture from any part of the room may add improved policies for the vacuum cleaning robot that were not originally possible.

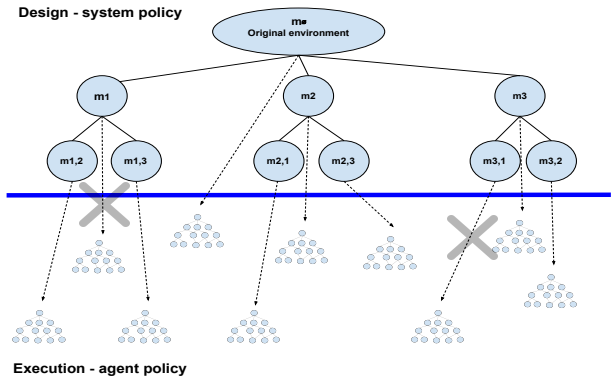

Figure 2: State space of a UMD problem

With the objective of finding efficient methods for the general UMD model, we observe that a UMD problem $\delta = \langle\mathcal{E}, \alpha, \sigma, \epsilon^0\rangle$ can be viewed as a tree comprising of two components (see Figure 2). The *design* component, at the top of the figure, describes the deterministic offline design process with nodes representing the different possibilities of modifying the environment. The *execution* component, at the bottom of the figure, represents the stochastic modified environments in which agents act.

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

When a design node is associated with an allowed modification (*i.e.*, $\Phi_\sigma(\vec{m}) = 1$) it is connected to a leaf node representing the environment $\epsilon_{\vec{m}}$ that results from applying the modification. To illustrate, invalid modification sequences are crossed out in Figure 2.

Algorithm 1 Best First Design (BFD)

BFD(δ, h)

```
1: create OPEN list for unexpanded nodes
2: n_cur = ⟨design, m⃗_∅⟩ (initial model)
3: while n_cur do
4:    if IsExecution(n_cur) then
5:       return n_cur.m⃗ (best modification found - exit)
6:    end if
7:    for each n_suc ∈ GetSuccessors(n_cur, δ) do
8:       put ⟨⟨design, n_suc.m⃗⟩, h(n_suc)⟩ in OPEN
9:    end for
10:   if Φ_σ(n_cur.m⃗) = 1 then
11:      put ⟨⟨execution, m⃗_new⟩, Usys(ε^0_{m⃗_new})⟩ in OPEN
12:   end if
13:   n_cur = ExtractMax(OPEN)
14: end while
15: return error
```

We exploit this structure and suggest an informed search in the space of allowed modifications, using heuristic estimations to guide the search more effectively by focusing attention on more promising redesign options. The *Best First Design* (BFD) algorithm (detailed in Algorithm 1) accepts as input a UMD model $\delta = \langle \mathcal{E}, \alpha, \sigma, \epsilon^0 \rangle$, and a heuristic function h. The algorithm starts by creating an OPEN priority queue (line 1) holding the front of unexpanded nodes. In line 2, n_{cur} is assigned the original model, which is represented by a flag $design$ and the empty modification sequence $\vec{m}_\emptyset$.

The iterative exploration of the currently most promising node in the OPEN queue is given in lines 3-14. If the current best node represents an execution model(indicated by the *execution* flag) the search ends successfully in line 5, returning the modification sequence associated with the node. Otherwise, the successor design nodes of the current node are generated by $GetSuccessors$ in line 7. Each successor sub-problem n_{suc} is placed in the OPEN list with its associated heuristic value $h(n_{suc})$ (line 8), to be discussed in detail next. In addition, if the modification sequence $n_{cur}.\vec{m}$ associated with the current node is valid according to Φ_σ, an execution node is generated and assigned a value that corresponds to the actual system value $\mathcal{U}sys(\epsilon^0_{\vec{m}_{new}})$ in the resulting environment (lines 10-12). The next node to explore is extracted from OPEN in line 13.

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

For optimality, we require the heuristic function h to be admissible. An admissible estimation of a design node n, formally given in Definition 5, is one that never underestimates $\mathcal{U}^{\max}(\delta)$, the maximal system's utility in the UMD problem δ represented by n.[1]

Definition 5 *Given a UMD model $\delta = \langle \mathcal{E}_\delta, \alpha_\delta, \sigma_\delta, \epsilon^0_\delta \rangle$, a heuristic function $h(\delta)$ is admissible if*

$$h(\delta) \geq \mathcal{U}^{\max}(\delta)$$

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

Theorem 1 *Given a UMD model δ and an admissible h, BFD(δ, h) returns $\vec{\mathrm{m}}^* \in \vec{\mathcal{M}}^*_\delta$.*

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

Admissible Heuristics for UMD

Using heuristic search promotes the need to develop informative heuristics that are relativity easy to compute. Informative admissible heuristics can be obtained by solving suitable simplifications of the input problem, which are typically achieved by relaxing various aspects of the problem at hand (Pearl 1984). This feature allows using these estimations as admissible heuristics. In the UMD case, a simplification of a model is one that is guaranteed to produce a model whose maximal system utility $\mathcal{U}^{max}(\delta)$ is at least as high as that of the real value. We let Δ represent the set of possible UMD models and define a simplification as follows.

Definition 6 *A function $f : \Delta \to \Delta$ is a* simplification *if*

$$\forall \delta, \delta' \in \Delta \text{ if } \delta' = f(\delta) \text{ then } \mathcal{U}^{max}(\delta) \leq \mathcal{U}^{max}(\delta').$$

A key issue to notice when developing heuristics for UMD is the overlap between the design process, used to maximize system utility in the resulting model and the simplification applied by the heuristic methods in order to ease the solution of a model. This overlap may limit the informative value of a heuristic estimation. For example, if a modification adds friction to specific area of the model and the applied simplification ignores the undesired outcome of move actions (which is slipping), the solution of the simplified environment may not reveal the value of applying the modification.

Definition 3 allows an arbitrary definition of system reward, making it necessary to consider the specific characteristic of each UMD setting when formulating simplifications. In particular, the distinction the UMD framework creates between agent and system utility means that the standard ways for simplifying environments do not necessarily hold for the general UMD case. As an example we show an approach that produces admissible estimates for ER-UMD setting, but is not valid for the *GRD* case.

Finding $\vec{m}^*$ for ER-UMD

In a ER-UMD setting agents and system share the same utility function. We exploit this feature both in formulating an admissible heuristic for the value of a modification sequence and in presenting a compilation that embeds the design process into the description of a planning problem that can be solved using any off-the-shelf solver.

[1] When system utility is expressed as cost, we require it to never overestimate the real cost.

The *simplified-environment* heuristic The solution of a UMD problem involves the solution of a new sub-model at every node. We suggest to perform a one time simplification of the original model as part of a preprocessing stage and use the simplified model to produce heuristic estimates for the design nodes of the search.

Accordingly, the *simplified-environment* heuristic, denoted h^{sim}, estimates the value of applying a modification $\vec{m}$ to a UMD setting using the value of applying $\vec{m}$ to a UMD setting with a simplified environment. Let $\mathcal{U}_\alpha(\epsilon)$ denote agent utility in ϵ when starting at $s_{0,\epsilon}$.

Definition 7 *Given a UMD model $\delta = \langle \mathcal{E}, \alpha, \sigma, \epsilon^0 \rangle$, a function $f : \mathcal{E} \to \mathcal{E}$ is an* environment simplification *if $\forall \epsilon, \epsilon' \in \mathcal{E}$ if $\epsilon' = f(\epsilon)$ then $\mathcal{U}_\alpha(\epsilon) \leq \mathcal{U}_\alpha(f(\epsilon))$.*

Given a UMD model $\delta = \langle \mathcal{E}, \alpha, \sigma, \epsilon^0 \rangle$ and a simplification function f, we let $f(\epsilon^0)$ represent the simplified environment that results from applying f to ϵ^0 and $\delta^{sim} = \langle \mathcal{E}, \alpha, \sigma, f(\epsilon^0) \rangle$ as the resulting UMD model. The *simplified-environment* heuristic estimates the value of applying a modification sequence $\vec{m}$ to δ by the optimal solution of applying $\vec{m}$ to δ^{sim}.

$$h^{sim}(\delta) \stackrel{def}{=} \mathcal{U}^{max}(\delta^{sim}_{\vec{m}}) \tag{3}$$

where $\delta^{sim}_{\vec{m}}$ represents the UMD model that results from applying modification $\vec{m}$ to $\epsilon^0_{\delta^{sim}}$.

To illustrate, consider Example 1. A simplified environment may be one where agent's slipping is ignored. The search applies modifications, such as moving furnitures, on the simplified model and uses the optimal solution of the simplified environment as an estimate of the value of applying the modifications in the original (slippery) setting.

The literature is rich with simplification approaches, including adding macro actions that can do more with the same cost, removing some action preconditions, eliminating the negative effects of actions (delete relaxation) or eliminating undesired outcomes of actions (Holte *et al.* 1996). While differing in the applied approach, common to all is that agent utility cannot decrease (cost cannot increase).

To be useful, we choose simplifications that are guaranteed to be easier to solve, providing valuable information used to direct the search more efficiently. In particular, we suggest applying the commonly used *all outcome determinization*(Yoon *et al.* 2007), which creates a deterministic action for each probabilistic outcome of every action. Using Definition 5 for admissible heuristics for UMD, Lemma 1 ensures the admissibility of the simplified-environment heuristic for ER-UMD using this approach.

Lemma 1 *Given a ER-UMD model $\langle \mathcal{E}, \alpha, \sigma, \epsilon_0 \rangle$, applying the* simplified-environment *heuristic with f implemented as an all outcome determinization function is admissible.*

The proof of Lemma 1, omitted for brevity, uses the observation that f only adds solutions with higher reward (lower cost) to a given problem (either before or after redesign). A similar reasoning can be applied to the other approaches discussed above, and the delete relaxation in particular.

The *simplified-environment* heuristic is not guaranteed to produce admissible estimates for the general UMD case, as shown below.

Lemma 2 *h^{sim} is non-admissible for* GRD.

The proof, omitted for brevity, uses a simplified environment to which the all outcome determinization was applied to show that the *wcd* in the simplified environment may overestimate the true *wcd* value.

The *simplified-environment* heuristic relies on the optimal solution of a simplified UMD model which can be performed using the *DesignComp* compilation presented next.

ER-UMD compialtion to planning For finding an optimal sequence of modifications to apply to an ER-UMD setting with a specified design budget, we suggest a compilation that embeds design into a planning problem. This is done by adding operators that modify the environment and making sure these modification actions are applicable only during an initialization stage that precedes the execution of agent policy. When initialization is complete, the agent acts in the optimized environment. This approach can be used by the *simplified-environment* heuristic, as well as a stand-alone approach for solving ER-UMD settings.

The ability to embed design in the search for an optimal policy relies on the alignment of agent and system utility assumed in the ER-UMD setting, making it inapplicable to the the *GRD* setting, where agent and system utility are not aligned. This alignment allows using any off the shelf optimal solver to find optimal modification sequence $\vec{m}^*$ to be applied while seeking the optimal agent policy.

The compilation approach is inspired by the technique of Göbelbecker *et al.* (2010) of coming up with good excuses for why there is no solution to a planning problem, an approach later extended to deal with various modification options in deterministic settings (Herzig *et al.* 2014; Menezes *et al.* 2012; Eiter *et al.* 2010). The compilation presented here extends this approach in three ways: by addressing stochastic environments rather than deterministic ones, by finding modifications to maximize the utility of the agent, rather than only moving from unsolvable to solvable and by embedding the support of a design budget.

The *DesignComp* compilation is formulated using the PPDDL notation (Younes and Littman 2004) to describe an infinite horizon discounted reward MDP, represented using a factored representation and defined by the tuple $\langle \mathcal{X}, S, A, s_0, \mathcal{R}, \gamma \rangle$ where the specification of a state $s \in S$ is a combination of values of several state variables $\mathcal{X}$ (Mausam 2012). The definition excludes the transition function which is embedded in the description of actions, each represented as a tuple of the form $\langle prec, \langle p_1, add_1, del_1 \rangle, \ldots, \langle p_m, add_m, del_m \rangle \rangle$ where $prec$ represents the preconditions of the action as a conjunction of literals that need to be true for the action to be applicable. The list $\langle p_1, add_1, del_1 \rangle, \ldots, \langle p_m, add_m, del_m \rangle$ are the probabilistic effects, where p_i is the probability of the i-th effect, add_i is the conjunction of positive literals that the effect adds to the state description and del_i is the conjunction of negative literals (positive literals that are set to false) added to the state description.

Corresponding to the structure depicted in Figure 2, the policy of the compiled planning problem has two stages: *design* - in which a budget is used to modify the system and

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

The compilation process supports two types of modifications: changing the initial state and changing the action set. Accordingly, the design actions set $A_{des} = A_{des\text{-}s_0} \cup A_{des\text{-}A}$ specifies two action types. Actions in $A_{des\text{-}s_0}$ change the initial state by setting the value of a state variable $x \in \mathcal{X}_{\mathcal{M}} \subseteq \mathcal{X}$ to true in the initial state. The set $A_{des\text{-}A}$ specifies a set of actions $A_{\mathcal{M}}$ that can be added to the environment. This is implemented by initially disabling all actions $A_{\mathcal{M}}$. Each action $a_{des-a} \in A_{des\text{-}A}$ applied during the design stage enables the execution of $a \in A_{\mathcal{M}}$ by setting its flag $enabled_a$ to true. All design actions have uniform cost.

The set $A_{exe} = A \cup A_{\mathcal{M}}$ includes the set of actions A from the original model and the set $A_{\mathcal{M}}$ of actions that can be enabled by the design process. In particular, we include in $A_{\mathcal{M}}$ actions that share the same structure as an action in A except for a modified probability distribution.

The budget B is implemented using a timer mechanism as in (Keren *et al.* a) which advances with the application of each design action. The timer limits the solution to include at most B design actions before applying a_{start}. Following is the definition of *DesignComp*.

Optimally solving the compiled problem P' yields an optimal policy $\pi^*_{P'}$ with two components, separated by the execution of a_{start}. The initialization component consists of a possibly empty sequence of deterministic design actions denoted by $\vec{m}_{P'}$, while the execution component represents the optimal policy in the modified environment. The set of optimal modification sequences of the original ER-UMD problem P is represented by $\vec{\mathcal{M}}^*_P$.

The next two propositions establish the correctness of the compilation. Proofs are omitted due to space constraints. We first argue that the expected reward in the compiled planning problem is exactly the expected reward accumulated in the optimal modified environment.

Lemma 3 *Given an ER-UMD problem P and an optimal modification sequence $\vec{\mathrm{m}}_{P'} \in \vec{\mathcal{M}}^*_P$*

$$V^*(s'_0) = V^*_{\epsilon_{\vec{\mathrm{m}}^*_P}}(s_{0,\epsilon_{\vec{\mathrm{m}}^*_P}})$$

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

Corollary 1 *Given an ER-UMD problem P and the compiled model P', $\vec{\mathrm{m}}_{P'} \in \vec{\mathcal{M}}^*_P$*

To ensure that the compilation not only respects the design budget B, but also minimizes design cost as much as possible, we assign a small cost c_d to design actions A_{des}. Note that if this cost is too high, it might lead the solver to omit design actions that improve utility by less than c_d. However, the loss of utility will be at most $c_d B$. Thus, by bounding the minimum improvement in utility from a modification, we can still ensure optimality.

Empirical Evaluation

Our evaluation aims at measuring the effect of a budget on the utility of a ER-UMD problem, as well as the performance of both optimal and approximate techniques for solving a ER-UMD problem **Datasets** We used six PPDDL domains from the probabilistic tracks of the sixth and eighth International Planning Competition[2] (IPPC06 and IPPC08) representing stochastic shortest path MDPs with uniform action cost: Box World (IPPC08/ BOX), Blocks World (IPPC08/ BLOCK), Exploding Blocks World (IPPC08/ EX-BLOCK), Triangle Tire (IPPC08/ TIRE) and Elevators (IPPC06/ ELEVATOR). For each domain, we created 10 simplified instances, considering only ones optimally solvable in their original unmodified formulation within a time bound of five minutes. For each domain we examined at least two possible modifications, including at least one that modifies the probability distribution. The specific modifications applied to each domain are specified in Table 2 where *change init* refers to modifications applied to the initial state and *probability change* to the probability function.

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

Table 2: Allowed modifications for each domain

Setup We optimally solved each problem using:
- Exhaustive exploration of all possible modifications (EX).
- Solution of the design *DesignComp* (DC) compilation.
- Execution of the BFD algorithm with the *simplified-environment* heuristic using the delete relaxation to simplify the UMD model and the *DesignComp* to optimally solve the simplified models (BFD).

We used a portfolio of 3 admissible heuristics:
- h_0 assigns 0 to all states and serving as a baseline for the assessing the value of more informative heuristics.
- h_{0^+} assigns 1 to all non-goal states and 0 otherwise.
- h_{MinMin} solves all outcome determinization using the zero heuristic (Bonet and Geffner 2005).

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

	B=1		B=2		B=3	
	solved	improved	solved	improved	solved	improved
BOX	8	0.279	8	0.42	7	0.44
BLOCK	6	0.207	3	0.24	3	0.24
EX-BLOCK	10	0.415	9	0.415	9	0.415
TIRE	9	0.44	8	0.511	6	0.537
ELEVATOR	9	0.22	7	0.24	0	NA

Table 3: Ratio of value improvement for optimal solvers

Results Separated by domain, Table 3 summarizes the ratio of expected cost improvement brought by the design process, as well as the number of solved instances for each budget value. With the exception of EX-BLOCK, reduction

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

	Box			Blocks			Ex. Blocks			Triangle Tire			Elevators		
	B=1	B=2	B=3	B=1	B=2	B=3	B=1	B=2	B=3	B=1	B=2	B=3	B=1	B=2	B=3
Ex-h_0	158.42(8)	264.79(7)	238.54(4)	50.54(6)	28.03(4)	348.93(2)	69.39(9)	161.702 (9)	250.70 (9)	32.95(9)	55.16(7)	270.335(6)	300.38 (8)	361.79 (5)	na
Ex-h_{0+}	159.09(8)	264.87(7)	236.47 (4)	50.51(6)	28.25(4)	347.34(2)	70.19(9)	170.87(9)	265.92(9)	32.99 (9)	55.45(7)	136.5(6)	299.56 (8)	360.92(5)	na
Ex-h_{MinMin}	158.92(8)	267.77(7)	235.59 (4)	50.78(6)	28.03(4)	348.22(2)	69.91 (9)	168.10(9)	292.20(9)	33.15(9)	55.03(7)	258.38(6)	301.6(8)	366.232 (5)	na
DC-h_0	163.9(8)	270.56(7)	241.5175(4)	50.71(6)	28.22(4)	354.515 (2)	68.40 (9)	153.12(9)	252.48(9)	33.26(9)	55.53(7)	269.71(6)	301.88 (8)	363.396(5)	na
DC-h_{0+}	70.70 (8)	92.08(8)	73.55 (4)	41.72(6)	17.39(4)	194.64(3)	38.73(9)	88.20(9)	134.91(9)	30.16(9)	51.1(8)	**136.5**(6)	236.21 (9)	260.98(5)	1504.65 (1)
DC-h_{MinMin}	221.40 (8)	332.70(7)	271.69 (4)	77.1 (6)	36.41(3)	363.48 (2)	**6.7(10)**	**30.18**(10)	**88.82(8)**	36.85(9)	88.82(8)	258.38(6)	192.6(9)	243.89(5)	1117.4 (1)
BFD-h_0	157.38 (8)	260.80(7)	234.31 (4)	50.35 (6)	28.07(4)	352.24(2)	69.54 (9)	153.9(9)	285.86(9)	32.99(9)	55.03(7)	267.64 (6)	302.63(8)	360.86(5)	na
BFD-h_{0+}	**68.21**(8)	**88.01**(8)	**70.25** (7)	**41.62** (6)	**17.16**(4)	**118.17**(3)	40.35 (9)	85.58(9)	160.87(9)	**29.51**(9)	**50.9**(8)	188.29(6)	238.28 (9)	258.64 (5)	1465.81 (1)
BFD-h_{MinMin}	216.39(8)	325.32(7)	265.94(4)	74.36 (6)	35.39(3)	354.85(2)	60.29 (9)	135.01 (9)	237.40(9)	36.85(9)	89.05(8)	256.25(6)	**176.62**(9)	**231.17**(5)	**1042.54**(1)

Table 1: Running time and number of instances solved for the optimal solvers

in expected cost increases with the budget increase, demonstrating the applicability of the UMD problem.

Table 1 compares solutions' performance. Each row represents a solver and heuristic pair. Results are separated by domain and budget, depicting the average running time for problems solved by all approaches for a given budget and the number of instances solved in parenthesis. The dominating approach for each row (indicating a domain and budget) is emphasized in bold. In all case, the use of informed search outperformed the exhaustive approach on all domains. However, the dominating heuristic approach varied between domains, and for TIRE also between budget allocation.

Conclusions and Discussion

We presented a general model for redesigning stochastic environments to maximize system utility. Utility functions for agents and system may be the same, aligned, or completely different. Extending earlier works, we show how two different models, one for goal recognition design, and the other for maximizing agent utility, can be considered as special cases of the general model. We then presented a general method for solving UMD problems using informed heuristic search. For settings where agents and system share the utility function, we presented an approach to produce admissible estimations and a compilation-based method that embeds design into the definition of a planning problem. Our empirical evaluation supports the feasibility of the approaches and shows substantial utility gain on all evaluated domains.

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

References

Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

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.

Thomas Eiter, Esra Erdem, Michael Fink, and Ján Senko. Updating action domain descriptions. *Artificial intelligence*, 2010.

Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. Coming up with good excuses: What to do when no plan can be found. *Cognitive Robotics*, (10081), 2010.

Andreas Herzig, Viviane Menezes, Leliane Nunes de Barros, and Renata Wassermann. On the revision of planning tasks. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, ECAI'14, 2014.

Robert C Holte, Maria B Perez, Robert M Zimmer, and Alan J MacDonald. Hierarchical a*: Searching abstraction hierarchies efficiently. In *AAAI/IAAI, Vol. 1*, pages 530–535. Citeseer, 1996.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design for non optimal agents. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2015)*.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design with non oservable actions. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI 2016)*.

Sarah Keren, Avigdor Gal, and Erez Karpas. Goal recognition design. In *ICAPS Conference Proceedings*, June 2014.

Sarah Keren, Avigdor Gal, and Erez Karpas. Privacy preserving plans in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2016)*, July 2016.

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

M Viviane Menezes, Leliane N de Barros, and Silvio do Lago Pereira. Planning task validation. In *Proc. of the ICAPS Workshop on Scheduling and Planning Applications*, 2012.

Nils J Nllsson. Principles of artificial intelligence. *TiogaSpringer Verlag. Palo Alto. Calif*, 1980.

Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

Tran Cao Son, Orkunt Sabuncu, Christian Schulz-Hanke, Torsten Schaub, and William Yeoh. Solving goal recognition design using asp. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, 2016.

Christabel Wayllace, Ping Hou, William Yeoh, and Tran Cao Son. Goal recognition design with stochastic agent action outcomes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2016)*, July 2016.

Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, pages 352–359, 2007.

Hakan LS. Younes and Michael L. Littman. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proc. International Planning Competition*, 2004.