Hindsight Optimization for Probabilistic Planning with Factored Actions

Murugeswari Issakkimuthu[a], Alan Fern[b], Roni Khardon[a], Prasad Tadepalli[b], Shan Xue[b]
[a]Department of Computer Science, Tufts University, Medford, MA 02155, USA
[b]School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA

Abstract

Inspired by the success of the satisfiability approach for deterministic planning, we propose a novel framework for on-line stochastic planning, by embedding the idea of hindsight optimization into a reduction to integer linear programming. In contrast to the previous work using reductions or hindsight optimization, our formulation is general purpose by working with domain specifications over factored state and action spaces, and by doing so is also scalable in principle to exponentially large action spaces. Our approach is competitive with state-of-the-art stochastic planners on challenging benchmark problems, and sometimes exceeds their performance especially in large action spaces.

1 Introduction

Many realistic probabilistic planning domains are naturally modeled as dynamic Bayesian networks (DBNs) over a set of state and action variables. DBNs allow for compact encoding of planning domains with large factored state spaces, factored action spaces allowing for concurrent actions, large stochastic outcome spaces for actions, and complex exogenous events. Recently, with the introduction of RDDL and related languages for probabilistic planning domains, specifying such domains compactly has become relatively easy, paving the way for the application of probabilistic planners to realistic problems.

Unfortunately, as we overview in Section 2, the current leading frameworks for probabilistic planning face serious scalability issues when encountering many of the above problem features. This motivates us to consider a new framework inspired by the success of the planning as satisfiability (Kautz and Selman 1992) for deterministic planning. The key idea of that work was to reduce deterministic planning problems to propositional satisfiability (SAT) problems and then apply state-of-the-art SAT solvers. Reduction-based approaches for probabilistic planning have received relatively little attention compared to SAT-based approaches for deterministic planning. Some exceptions include reductions to stochastic satisfiability (Majercik and Littman 1998; 2003), stochastic CSPs (Hyafil and Bacchus 2004), and weighted model counting (Domshlak and Hoffmann 2006). Those approaches, however, have either not demonstrated scalability and/or have been restricted to finding open-loop plans. Furthermore, these approaches have so far only been developed for concrete action spaces and hence are not applicable to factored action spaces. The question then is whether one can develop a reduction-based approach, which is both general and able to handle factored action spaces.

The main contribution of this paper is to propose a framework and conduct a first investigation into solving domain-independent probabilistic planning for factored domains through integer linear programming (ILP), allowing for the exploitation of state-of-the-art ILP solvers. To achieve this we turn to Hindsight Optimization (HOP) (Chang, Givan, and Chong 2000; Chong, Givan, and Chang 2000) as a framework for approximately reducing probabilistic planning to ILP. HOP is a heuristic approach for online action selection based on the idea of computing optimistic bounds on action values via problem determinization, allowing for the use of deterministic solvers for probabilistic planning. While it is known that HOP's optimistic nature can lead to poor performance in certain types of domains, it has been successfully applied across a diverse set of challenging probabilistic planning domains (Chang, Givan, and Chong 2000; Chong, Givan, and Chang 2000; Wu, Chong, and Givan 2002; Yoon et al. 2008; 2010; Eyerich, Keller, and Helmert 2010; Hubbe et al. 2012; Xue, Fern, and Sheldon 2014). Unfortunately, existing HOP approaches either scale linearly with the number of actions, which is impractical for many factored-action problems, or are domain specific. In this paper, we show a general approach for translating a DBN-style factored planning problem to an integer linear program (ILP) that captures the HOP objective. Moreover, the formulation is naturally factored and it allows the optimization over large action spaces.

Our reduction to ILP translates the planning problem into one large ILP, which gives rise to a number of questions about feasibility. Given that integer programming is hard in general, could we even expect the ILPs to be solved within a reasonable amount of time for typical benchmarks? What if we are not able to optimize the ILP and are forced to use sub-optimal solutions obtained within the allotted time? Can this approach compete with state-of-the-art domain-independent planners across a variety of domains? Does the solution quality suffer due to the inherent optimism of HOP?

The paper studies these questions experimentally by evaluating the algorithm and comparing it to the state-of-the-art system PROST (Keller and Eyerich 2012). As our experiments show, in many cases our approach provides competitive or superior performance. However, the size of the resulting ILP and its solvability plays an important role in the success of this method, and improving encoding size, suitability to ILP solvers, as well as improving the overall optimization scheme are important problems for future work.

2 Background

2.1 Factored Markov Decision Processes

This paper assumes some familiarity with the basic framework of Markov Decision Processes (MDPs) (Puterman 2005). We are concerned with finite-horizon online planning in stochastic domains modeled as factored MDPs. A MDP is a 4-tuple $\langle S, A, T, R \rangle$ where S is a finite set of states, A is a finite set of actions, T is a state transition function and R is a reward function. The transition function $T(s, a, s')$ gives the probability of reaching state s' by taking action a in state s and the reward function $R(s)$ associates a numeric reward with state s. Note that we follow the standard formulation of these settings that does not include a discount factor but this is easy to incorporate into our algorithms if needed. The standard solution of a MDP is a policy which is a mapping of states to actions. In our setting of online planning the focus is on efficiently computing a high-quality action for the current state s after each state transition.

In a factored MDP, the state space S is described by a finite set of binary variables $(x_1, x_2, \ldots, x_N)$ and the action space is described by a finite set of binary variables $(a_1, a_2, \ldots, a_n)$. In some cases, the state and action spaces are also subject to state and action constraints that eliminate certain combinations of values as illegal states or actions. We assume in this work that the transition function T is compactly described as a Dynamic Bayesian Network (DBN) which specifies the probability distribution over each state variable x_i in the next time step, denoted x'_i, given the values of a subset of the state and action variables parents(x'_i). In particular, we have $T(s, a, s') = \prod_i Pr(x'_i|\text{parents}(x'_i))$. The conditional probability tables (CPTs) $Pr(x'_i|\text{parents}(x'_i))$ are often represented via a structured representation such as a decision tree, rule set, or algebraic decision diagram. We also assume a similarly compact reward function description in terms of the state variables.

2.2 Prior Approaches

There are two main approaches for solving factored MDPs. First, symbolic dynamic programming (SDP) algorithms attempt to compute symbolic representations of policies and value functions over the entire state-space. SDP has been developed for concrete action spaces (Hoey et al. 1999), factored action spaces (Raghavan et al. 2012; 2013), and extended for using information from the current state to focus the search and reduce complexity (Feng and Hansen 2002; Feng, Hansen, and Zilberstein 2003). Unfortunately, these planners have not yet exhibited scalability to large problems, in part due to their goal of computing policies over the entire reachable portion of the state space.

The second type of approach is online tree search, where planning is done for only the current state by constructing a look-ahead search tree rooted at that state in order to estimate action values. Such tree search algorithms are currently the state-of-the-art in terms of empirical performance on certain benchmarks and include planners such as PROST (Keller and Eyerich 2012), Glutton (Kolobov et al. 2012), and Anytime AO* (Bonet and Geffner 2012), among others. Even these planners, however, exhibit serious scalability issues for many domains. In particular, the planners treat each action as atomic, which dramatically increases their run time for factored action domains. This, in combination with large stochastic branching factors, due to exogenous events, severely limits the achievable search depths. While this can be mitigated to some extent by pruning mechanisms or heuristics, applicability of search methods to large factored action spaces remains a challenge.

2.3 Hindsight Optimization

Given the inherent challenges of the above frameworks, it is important to consider alternative frameworks that may offer complementary strengths. In this work, we follow the online planning strategy of the tree-search approaches, but employ hindsight optimization (HOP) to compute actions.

The main idea of HOP is to identify the best action in state s, by calculating an optimistic estimate of $Q(s, a)$ for each a and choosing the action with highest estimate. To achieve this, for each a, the HOP algorithm (1) takes multiple samples of s' from the transition function $T(s, a, s')$, (2) randomly selects a "future" from s' by determinizing all future potential transitions, and (3) evaluates "the value of s' with hindsight" by solving the resulting deterministic planning problem. Previous work represented and evaluated each future separately, and explicitly averaged the value of each a through its successor states s'. This yields an optimistic estimate for $Q(s, a)$ which can be used in action selection.

The advantage of HOP is its ability to perform deep search in each of the independent determinized futures. This contrasts with the above tree-search approaches where depth is a significant limiting factor. On the other hand, unlike the tree-search approaches, in general, HOP is not guaranteed to select an optimal action even when given an exhaustive set of futures. HOP constructs a plan based on presumed outcomes of all future actions, which makes it inherently optimistic and leads to suboptimal plans in general (Yoon et al. 2008). In spite of this significant limitation, HOP has been successfully applied to a variety of stochastic domains with small action spaces(Chang, Givan, and Chong 2000; Chong, Givan, and Chang 2000; Wu, Chong, and Givan 2002; Yoon et al. 2008; 2010; Hubbe et al. 2012) and domain-independent systems have been developed (Yoon et al. 2008; 2010). However, the complexity of HOP in previous work scales linearly with the number of actions, which is not feasible for large factored action spaces. The question then is whether a more efficient version of HOP can be developed.

It is interesting to note that the HOP solution is a simplified form of the sample average approximation (SAA) algorithm (Kleywegt, Shapiro, and Homem-de Mello 2002) from stochastic optimization literature, which is closely related to the Pegasus algorithm (Ng and Jordan 2000) for POMDPs. Like HOP, the SAA solution draws multiple deterministic futures in order to get an approximate solution of the original problem. However, unlike HOP, SAA does not allow for independent solutions of those futures. As a result SAA, is more demanding computationally but it can be shown to converge to the optimal policy under some conditions. The next section shows how HOP can be used in factored spaces. Extensions of these ideas to SAA are left to future work.

3 Hindsight Optimization via ILP

In order to scale HOP to factored actions, we turn to a recent idea that was developed in the context of a bird conservation problem (Kumar, Wu, and Zilberstein 2012; Xue, Fern, and Sheldon 2014) where the corresponding domain specific problem is translated into a mixed integer program (MIP). In particular, the work of (Xue, Fern, and Sheldon 2014) showed how one can translate the HOP objective in that problem into a MIP that is expressed directly over state and action variables. The MIP is then solved to produce the HOP action for the current state. This work showed that at least for specific cases, the HOP solution can be computed for exponentially large factored-action spaces. The question then is whether this approach can be generalized into a domain-independent planning method.

In this section, we describe such a domain-independent HOP approach that translates the HOP objective to an integer linear program (ILP) that naturally factors over both state and action variables. In this way, the size of the ILPs will grow reasonably with the number of action variables. This provides the potential for computing HOP actions in exponentially large action spaces within a reasonable time, depending on the effectiveness of the ILP solver.

3.1 DBN Domain Representation

We assume that we have as input a compact representation of the DBN for the domain. This includes a description of the transition function, the reward function, and possibly state and action constraints. Previous work has used trees or decision diagrams for such representations (Boutilier, Dean, and Hanks 1999). In our experiments we use the Relational Dynamic Influence Diagram Language (RDDL) (Sanner 2010) as the high level specification language for stochastic planning. RDDL is the current standard used in recent planning competitions. It can be translated into a ground representation in various formats, and a translator into decision diagrams is provided with the distribution of the simulator (Sanner 2010). Here we use as input a similar translation, developed in (Raghavan et al. 2012), where conditional probability tables (CPT) and the reward function are given as decision trees.

In the following, we first explain how one can use trees and then show how in some cases it is beneficial to use a more compact set of rules. Nodes in the tree representation are labeled with propositional variables and edges are labeled with truth values. A path from the root to a leaf in the tree captures a conjunction of state and action literals on that path; the conjunctions corresponding to different paths are mutually exclusive and they therefore define a partition of the state and action space. The leaves of the reward function tree are real values representing the reward for the corresponding partition. For state variables, we have a separate tree for each x'_i capturing $Pr(x'_i = 1|\text{parents}(x'_i))$ where the parents include state and action variables. Leaves in these trees are numbers in $[0, 1]$ representing the conditional probabilities. For domains with state-action constraints the leaves in the constraint tree are labeled with binary values $\{0, 1\}$ where 0 identifies illegal combinations of variables.

In this paper we consider all paths or state-action partitions in the trees and translate them to linear constraints. We note that the same ideas can be applied to ADDs in a straightforward way although we have not implemented this in our system. In some cases the number of paths, at least in automatically derived translations, is unnecessarily large. We therefore work with a slightly improved representation that builds on two intuitions. First, given a tree with binary leaves it is sufficient to represent either the paths leading to 0 or the paths leading 1, by taking the default complement value when the paths are not satisfied. This allows us to shrink the representation significantly in some cases where one set of paths is much smaller. In addition, in this case the paths considered need not be mutually exclusive (they all lead to 0 or all lead to 1) and can therefore be more compact. The second intuition is that in many domains the transition is given as a set of conditions for setting the variables to true (false) with corresponding probabilities but with a default value false (true) if these conditions are not satisfied. For example, $Pr(x'_1 = 1|x_1, x_2, x_3)$ might be specified as "$x_1 \wedge x_2 \Rightarrow p = 0.7$; $x_1 \wedge \overline{x_2} \Rightarrow p = 0.4$; otherwise p=0". In this case the translation can be simplified using the combination of rules and default values. We explain the details in the context of the ILP formulation below.

For the domains in our experiments we were able to use direct translation from RDDL into trees for some of the domains, but for domains where the trees are needlessly large we provided such a rule-based representation.

3.2 ILP Formulation

Recall that for HOP we need to sample potential "futures" in advance while leaving the choice of actions free to be determined by the planner. In order to do this, one has to predetermine the outcome of any probabilistic action in any time step (where the same action may be determinized with different outcomes at different steps). For the ILP formulation we have to explicitly represent the state and action variables in all futures and all time steps. Once this is done, we can constrain the transition function in each future to agree with the predetermined outcomes in each step. To complete the translation we represent the objective of finding the HOP action that maximizes the average value over all futures into the objective function of the ILP. The rest of this section explains this idea in more detail.

We start by introducing the notation. As discussed above, we are given a problem instance translated into a set of trees (or rule sets) $\{T_1, T_2, ..., T_N, R, C\}$, where $\{T_1, T_2, ..., T_N\}$

are transition function trees, R is the reward function tree, and C is the constraint function. Let $\{x_1, x_2, ..., x_N\}$ be the set of binary state variables, $\{a_1, a_2, ..., a_n\}$ be the set of binary action variables, m be the number of futures and h be the look-ahead (horizon) value. We use t to denote the time step and k to represent the index of the future/sample. Let $x^k_{i,t}$ and $a^k_{j,t}$ be the copies of x_i and a_j respectively for future k and time step t. Then $X^k_t = (x^k_{1,t}, x^k_{2,t}, ..., x^k_{N,t})$ denotes the state and $A^k_t = (a^k_{1,t}, ..., a^k_{n,t})$ denotes the action at time step t in the k^{th} future.

To facilitate the presentation we describe the construction of the ILP in several logically distinct parts. We start by explaining how to determinize the transition function into different futures. We then show how the futures are linked to capture the HOP action selection constraint, and how the HOP criterion can be embedded into the ILP objective. We then describe a few additional portions of the translation.

Preliminaries: Integer linear programs can easily capture logical constraints over binary variables. We recall these facts here and mostly use logical constraints in the remainder of the construction. In particular logical AND [$z = x_1 \wedge x_2 \wedge ... \wedge x_n$] is captured by $nz \leq (x_1 + x_2 + ... + x_n) \leq (n-1) + z$, logical OR [$z = x_1 \vee x_2 \vee ... \vee x_n$] is captured by $z \leq (x_1 + x_2 + ... + x_n) \leq nz$, and negation [$z = \neg x_1$] is captured by $z = 1 - x$ where in all cases we have $z \in \{0, 1\}$.

Determinizing Future Trajectories: We start by explaining how one can translate the tree representation into ILP constraints and then improve this for using rules.

In order to determinize the transition function we create $m \times h$ determinized versions of each tree, where $\{D^k_{i,t}\}$ is the determinization of T_i, by converting the probabilities at the leaves to binary values using a different random number at each leaf, i.e., the leaf is set to $true$ if the random number in [0, 1) is less than its probability value and $false$ otherwise.

Let $U_{i,p}$ be the set of positive state literals, $V_{i,p}$ be the set of state variables appearing in negative literals, $B_{i,p}$ be the set of positive action literals and $C_{i,p}$ be the set of action variables in negative action literals along the root-to-leaf path #p of T_i. Let $y^k_{i,p,t}$ be the binary value at the leaf of path #p in the determinized version $D^k_{i,t}$ of T_i. The state variable $x^k_{i,t+1}$ is constrained to take the leaf-value of the path in $D^k_{i,t}$ which is satisfied by the state $X^k_t = (x^k_{1,t}, x^k_{2,t}, ..., x^k_{N,t})$ and action $A^k_t = (a^k_{1,t}, ..., a^k_{n,t})$. We let $f^k_{i,p,t}$ denote the conjunction of all constraints in the path #p of the determinized version $D^k_{i,t}$ of T_i. The transition constraints for $x^k_{i,t+1}$ are

$$x^k_{i,t+1} = \begin{cases} y^k_{i,p,t} & \text{if } f^k_{i,p,t} = 1 \\ \text{unconstrained} & \text{if } f^k_{i,p,t} = 0 \end{cases}$$

which can expressed as

$$\begin{array}{ll} \neg f^k_{i,p,t} \vee x^k_{i,t+1} = 1 & \text{if } y^k_{i,p,t} = 1 \\ \neg f^k_{i,p,t} \vee \neg x^k_{i,t+1} = 1 & \text{if } y^k_{i,p,t} = 0 \end{array}$$

where

$$u^k_{i,p,t} = \wedge_{x_i \in U_{i,p}}\, x^k_{i,t}$$
$$v^k_{i,p,t} = \wedge_{x_i \in V_{i,p}}\, (1 - x^k_{i,t})$$
$$z^k_{i,p,t} = u^k_{i,p,t} \wedge v^k_{i,p,t}$$

$$b^k_{i,p,t} = \wedge_{a_j \in B_{i,p}}\, a^k_{j,t}$$
$$c^k_{i,p,t} = \wedge_{a_j \in C_{i,p}}\, (1 - a^k_{j,t})$$
$$g^k_{i,p,t} = b^k_{i,p,t} \wedge c^k_{i,p,t}$$

$$f^k_{i,p,t} = z^k_{i,p,t} \wedge g^k_{i,p,t}$$

Notice that the conditioning on the value of $y^k_{i,p,t}$ in the formula above is done at compile time and is not part of the resulting constraint. In particular, depending on the value of $y^k_{i,p,t}$ the formula for $f^k_{i,p,t}$ can be directly substituted in the applicable constraint. The intermediate variables $u^k_{i,p,t}$, $v^k_{i,p,t}$, $z^k_{i,p,t}$, $b^k_{i,p,t}$, $c^k_{i,p,t}$, $g^k_{i,p,t}$ and $f^k_{i,p,t}$ are introduced here for readability. In our implementation we replace them by the conjunctions directly as input to the ILP solver. This completes the description of the basic translation of trees.

To illustrate the construction, consider a hypothetical path $x_1 \wedge \overline{x_2} \wedge a_2$ leading to a leaf with value $y = 1$, as part of the transition function for say x_3 at time t. Then f represents the conjunction of $x_{1,t}$, $(1 - x_{2,t})$, and $a_{2,t}$ using linear inequalities. Then because we are in the case $y = 1$ we can write $(1 - f) + x_{3,t+1} = 1$. This captures the corresponding condition for $x_{3,t+1}$.

As a first improvement over the basic tree translation, note that once a tree is determinized its leaves are binary and , as explained in Section 3.1, we can represent explicitly either the paths to leaves with value 1 or paths to leaves with value 0. By capturing the logical OR of these paths we can set the variable to the opposite value when none of the paths is followed. We next discuss how this can be further improved with rule representations.

For the rule representation we allow either a set of rules sharing the same binary outcome and an opposing default value, or a set of mutually exclusive rules with probabilities, and a corresponding default value. The first case is easy to handle exactly as in the previous paragraph. For the second case, we must first determinize the rule set, which we do by drawing a different random number for each rule at each time step. This generates a determinized rule set analogous to $D^k_{i,t}$. To make this more concrete consider a rule set with 3 rules and a default value of 0: [$R_1 \to 0.7$; $R_2 \to 0.3$, $R_3 \to 0.9$; otherwise 0], and determinize it to get [$R_1 \to 1$; $R_2 \to 0$, $R_3 \to 1$; otherwise 0]. In this case we take only the rules leading to 1 (the value opposite to the default value), that is R_1 and R_3 in the example, represent their disjunction to force a 1 value of the next state variable, and forcing a 0 value when the disjunction does not hold.

Capturing the HOP action constraint: The only requirement for HOP is that all futures use the same action in the first step. In the translation above, variables from different futures are distinct. To achieve the HOP constraint we simply need to unify the variables and actions corresponding to the first state. Let $I \subseteq X$ be the subset of variables *true* in the initial state. The requirement can be enforced as follows:

$$\begin{array}{lll} x^k_{i,0} = 1 & \forall x_i \in I & \\ x^k_{i,0} = 0 & \forall x_i \in X - I & \forall k = 1..m \end{array}$$

$$a^k_{j,0} = a^{k+1}_{j,0} \qquad \forall j = 1..n, k = 1..m-1$$

Capturing the HOP quality Criterion: The HOP objective is to maximize the sum of rewards obtained in all the t time steps averaged over k futures. This can be captured directly in the objective function of the ILP as follows.

$$\text{Min } Z(A) = -\frac{1}{m}\sum_{k=1}^{m}\sum_{t=0}^{h}\sum_{p=1}^{L}\left(r_p \times w^k_{p,t}\right)$$

where r_p is the reward at the leaf of path #p of the reward tree, L is the number of leaves of the reward tree, and $w^k_{p,t}$ is an intermediate variable that represents the path constraint of #p, defined as

$$w^k_{p,t} = u^k_{p,t} \wedge v^k_{p,t}$$
$$u^k_{p,t} = \wedge_{x_i \in U_p} x^k_{i,t}$$
$$v^k_{p,t} = \wedge_{x_i \in V_p} (1 - x^k_{i,t})$$

U_p is the set of positive literals and V_p is the set of variables appearing in negative literals along path #p of R and $w^k_{p,t}$ is set to 1 if the conjunction of the literals along path #p is satisfied for $X^k_t = (x^k_{1,t}, x^k_{2,t}, ..., x^k_{N,t})$ and $A^k_t = (a^k_{1,t}, ..., a^k_{n,t})$ and 0 otherwise.

State Action Constraints and Concurrency Constraints: As mentioned above, in some domains we are given an additional tree C capturing legal and illegal combinations of state and actions variables. This is translated along similar lines as above. We can represent the disjunction of all paths to leaves that have value 1 as an intermediate variable, and then require that this variable is equal to 1. In addition, for our experiments we need to handle one further RDDL construct, the concurrency constraints, which is orthogonal to the DBN formulation, but has been used to simplify domain specifications in previous work, for example in planning competitions. These have been used to limit the number of parallel actions in any time step to some constant c. This can be expressed as

$$\sum_{j=1}^{n} a^k_{j,t} \leq c \qquad \forall t = 0..h, k = 1..m$$

Solving the hindsight ILP: We have used IBM's optimization software CPLEX to solve the ILPs. We create a new ILP at each decision point with the current state as the initial state using a set of newly determinized futures. Our construction mimics the HOP procedure, although algorithmically it is quite different. Previous work on HOP enumerates all actions, estimates their values separately by averaging values of multiple determinations for each action, and finally picks the action with the highest estimate. Instead of this, the ILP formulation, if optimally solved, provides an action which maximizes the average value over all determinized futures. This value is equal to the value in the simple HOP formulation, although several actions might be tied. We therefore have that:

Proposition 1 *If the simple HOP algorithm and the ILP-HOP algorithm use the same determined futures then the optimal solution of the ILP yields the same action and value as the simple HOP algorithm assuming a fixed common tie breaking order over actions.*

4 Experiments

We evaluate ILP-HOP on six different domains comparing to PROST (Keller and Eyerich 2012) the winner of the probabilistic track of IPPC-2011. Five of the domains — Sysadmin, Game of life, Traffic control, Elevators, and Navigation — are from IPPC-2011. One domain, Copycat, is new and has been created to highlight the potential of ILP-HOP.

We next explain the setup for ILP-HOP, for PROST, and for the experiments in general. The time taken by CPLEX to solve the ILP varies according to its size and it does not always complete the optimization within the given time bound. When this happens we use the best feasible solution found within the time-limit. The two main parameters for ILP-HOP are the number of futures and the planning horizon used to create the ILP. For the number of futures, we ran experiments with the values 1, 3, 5, 7, 10. In principle, one can tune the horizon automatically for each domain by running a simulation and testing performance. However, in this paper we used our understanding of each domain to select an appropriate horizon as described below. This is reasonable as we are evaluating the potential of the method for the first time and identifying an optimal horizon automatically is an orthogonal research issue. On the other hand, in order to provide a fair comparison with PROST, which attempts to automatically select planning horizons, we ran two versions of PROST. The first was the default version that selected its own problem horizon. The second was a version of PROST that we provided with the same horizon information given to ILP-HOP, which is used by PROST for its search and its Q-value initialization (heuristic computation). The reported value for PROST is the best of the two systems, noting that neither version was consistently better.

The experimental setup is as follows. In our main set of experiments, all planners are allowed up to 180 seconds per step to select an action as this gives sufficient time for the optimization in each step. We also consider 1 second per step in order to analyze the effect of sub-optimal ILP solutions on performance. For each problem in a domain we averaged the total reward achieved over 10 runs, except for Elevator, where 30 runs were used due to high variance. Each problem was run for a horizon of 40 steps. We report results in Table 1 and Figure 1 where Table 1 provides detailed results for all problems for a fixed horizon of 5 futures, and Figure 1 focuses on specific problems and varies the number of futures. Table 1 reports average reward achieved as well as the % of ILPs solved to optimality throughout all problem runs (400 ILPs, one for each step of each run) and the average time taken to solve the ILPs at each time step.

4.1 HOP Friendly Domain – Copycat

We first describe results for a new domain Copycat that we created to exhibit the potential benefits of HOP. In particular, this domain is parameterized so as to flexibly scale the number of action factors, the amount of exogenous stochasticity, and the lookahead depth required to uncover the optimal reward sequences.

Description. There are $N = n + d$ binary state variables $\{x_1, x_2, ..., x_n, y_1, y_2, ..., y_d\}$ and n binary action variables

Table 1: Average reward results and CPLEX solver statistics for ILP-HOP (5 futures) and PROST given 180 seconds and 1 second per time step. Average rewards are averaged over 10 runs for all problems, except Elevator which uses 30 runs. The CPLEX solver statistics give the % of ILPs solved optimally during the runs of each problems and the average time required by the solver to solve the ILP at each timestep

Average Reward (180 Seconds per Step)

Domain	Problem-2		Problem-3		Problem-5		Problem-6		Problem-8		Problem-10	
	HOP	Prost	HOP	Prost	HOP	Prost	HOP	Prost	HOP	Prost	HOP	Prost
Copycat	29.2	5.8	12.3	0.0	29.4	0.0	9.1	0.0				
Sysadmin	349.9	355.9	707.0	714.7	1057.2	1056.7	1041.9	1037.7	1388.7	1300.3	1510.2	1272.1
Game of life	176.0	186.5	158.1	172.5	352.3	297.1	290.3	293.5	527.7	411.5	723.6	578.9
Traffic control	-13.0	-9.6	-10.8	-8.5	-41.1	-37.6	-66.9	-55.2	-48.1	-49.9	-93.9	-115.5
Elevators	-26.2	-19.2	-61.9	-52.4	-57.5	-49.1	-71.9	-65.6	-77.6	-74.1	-75.3	-72.4

CPLEX Solver Statistics (180 Seconds per Step)

	ILP%	Time	ILP%	Time	ILP%	Time	ILP%	Time	ILP%	Time	ILP%	Time
Copycat	100	1.6s	100	8.5s	100	3.8s	100	27.4s				
Sysadmin	100	0.3s	100	0.4s	100	0.5s	100	1.7s	100	1.8s	100	5.6s
Game of life	100	2.9s	100	2.8s	100	8.6s	100	10.7s	99.25	24.8s	100	20.1s
Traffic control	100	0.6s	100	1.3s	100	2.5s	100	13.8s	100	21.1s	98.25	45.5s
Elevators	99.33	12.8s	97.42	21.1s	70.92	89.7s	61.33	109.6s	21.83	165.7s	86.75	59.6s

Average Reward (1 Second per Step)

	HOP	Prost	HOP	Prost	HOP	Prost	HOP	Prost	HOP	Prost	HOP	Prost
Copycat	25.5	0.2	0.0	0.0	14.3	0.0	0.0	0.0				
Sysadmin	348.4	355.9	709.4	695.5	1059.1	1043.8	1035.0	984.5	1390.5	1380.5	1420.4	1364.7
Game of life	166.3	189.1	155.2	173.4	333.5	279.1	259.8	269.2	369.8	405.2	635.4	506.2
Traffic control	-12.8	-9.9	-12.0	-21.8	-95.3	-49.0	-97.1	-71.5	-117.7	-70.3	-341.8	-106.4
Elevators	-47.2	-18.3	-71.3	-56.7	-111.4	-52.9	-132.6	-71.3	-151.0	-72.6	-126.4	-72.9

CPLEX Solver Statistics (1 Second per Step)

	ILP%	Time	ILP%	Time	ILP%	Time	ILP%	Time	ILP%	Time	ILP%	Time
Copycat	97.5	0.2s	30.75	0.9s	51.75	0.6s	95.6	0.28s				
Sysadmin	95.75	0.2s	99.75	0.1s	98	0.3s	94	0.3s	93.75	0.4s	57.75	0.8s
Game of life	45.75	1.2s	46.50	1.1s	31.75	1.1s	5	1.2s	4	1.4s	38.25	1.1s
Traffic control	89.5	1.1s	65.5	1.3s	17.25	1.4s	3.5	1.1s	0	1.2s	0	1.1s
Elevators	10.42	1.2s	0	1.2s	0	1.1s	0	1.1s	0	1.1s	0.58	1.2s

$\{a_i, a_2, ..., a_n\}$ in a problem instance. The start state has $y_1 = y_2 = ... = y_d = 0$ and the goal is to get to a state in which $y_d = 1$. From any state, the action that matches the x-part of the state is the only one that causes any progress towards a goal state. This action sets the leading unset y-bit to 1 and has a success probability 0.49. Once set, a y-bit remains set always. All incorrect actions leave the y-bits unchanged and all the actions cause the x-bits to change randomly. The optimal policy for any problem is to copy the x-part of the state and hence the name *Copycat*. The value of d determines how far away a goal state is from the initial state and the value of n determines the size of the action space and the stochastic branching factor. The size of the state space depends on both n and d. We tested the planners on four problem instances with all four combinations of 5 and 10 for n and d with a horizon value of $d + 6$.

Results. From Table 1 we see that PROST is only able to get non-zero reward on the smallest problem with $n = d = 5$. In general, this domain will break most tree search methods, unless action pruning and heuristic mechanisms provide strong guidance, which is not the case for PROST here. On the other hand, ILP-HOP is able to achieve non-trivial reward on all problems. This is due to the fact that the constructed ILP is able to solve the problem globally. We also see from Table 1 that these ILPs are relatively easy for CPLEX in that it is able to always produce optimal solutions quite quickly, with the largest problem instance requiring less than 30 seconds per step and the smallest instance requiring less than 2 seconds.

4.2 HOP Unfriendly Domain – Navigation

This domain from IPPC-2011 models the movement of a robot in a grid from a start cell to a destination cell. The problems are characterized by having a relatively long, but safe, path to the goal and also many shorter, but riskier paths, to the goal. Along risky paths there is always some probability that taking a step will kill the robot and destroy any chances of getting to the goal. Further, once the robot enters into one of the risky locations there is no way to go back to safe locations. The optimal strategy is usually to take the long and safe path, though some amounts of risk can be tolerated in expectation depending on the problem parameters.

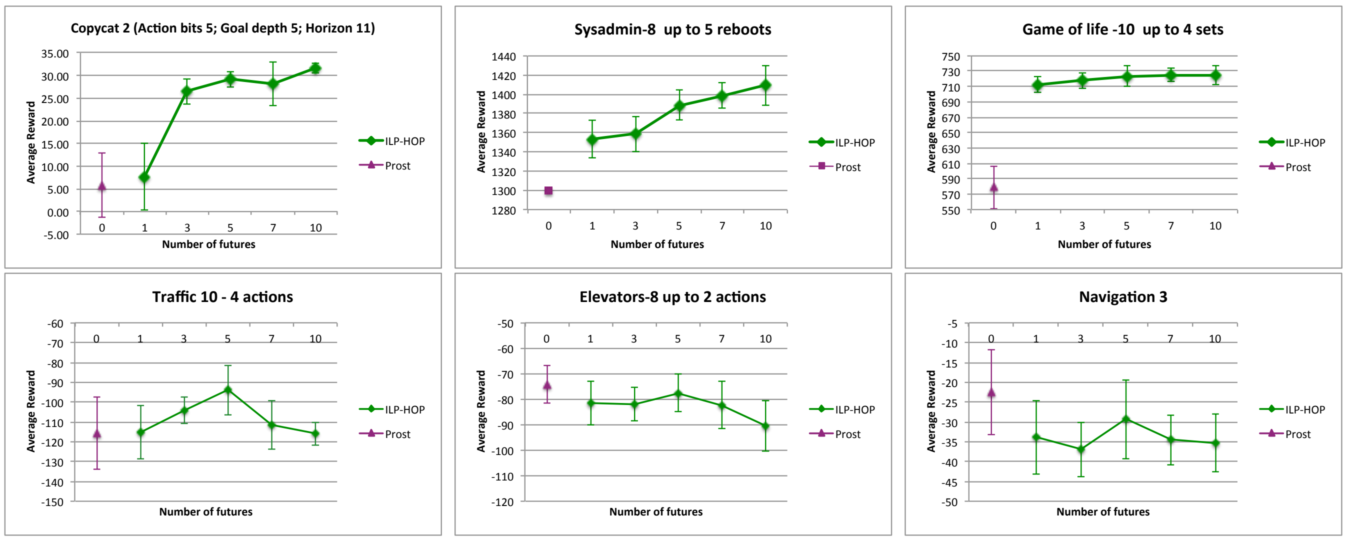

Figure 1: Performance of ILP-HOP on a single problem from each domain for different numbers of futures given 180 seconds per decision. The performance of PROST given 180 seconds per decision is also given.

We have included this domain because it is an example of an existing benchmark with the type of structure that can cause HOP to perform poorly. In particular, in this domain, HOP will often choose one of the riskier options (go to a risky cell), depending on the precise details of the domain, and once that choice is made the domain does not allow returning to safe cells. To understand why HOP can behave in this way consider a decision point where the robot is at a grid location and faced with the decision to take a step on the safe path, the risky path, or do nothing. HOP will first sample a set of futures that will each indicate, for each location in the possible paths to the goal and for each time step in the future, whether the robot would be killed at that location and time step. For each future, then, there is often a risky path to the goal that looks safe in hindsight. Further, when evaluating whether to take a step onto a risky location, the hindsight value from that location averaged across the futures will often be greater than the value of taking a step along the safe path. Thus, HOP is likely to choose to move to a risky location at some point during an episode.

Our experiments indeed show that ILP-HOP fails to perform well in this domain. An example of ILP-HOP's performance for different numbers of futures on a relatively small problem is shown in Figure 1. The performance across all numbers of futures is close to the minimum performance of -40. Prost is able to do significantly better on this problem.

4.3 Benchmark Domains

We now consider performance on our four remaining IPPC-2011 benchmarks — Sysadmin, Game of Life, Traffic, and Elevator — for which both PROST and ILP-HOP achieve non-trivial performance. For each domain, we use problem instances 2, 3, 5, 6, 8, and 10. Here we focus on performance achieved at 180 seconds per step.

Sysadmin and Game of Life. Both of these domains have the potential for combinatorial action spaces due to concurrency (e.g. rebooting n of c computers). The IPPC-2011 instances, however, did not allow for concurrency and hence the action spaces were relatively small. In order to investigate performance for combinatorial actions spaces we increased the allowed concurrency in each domain to be 4, 3, 5, 3, 5 and 4 respectively for the six problems. So for example, problem 8 from Sysadmin allows for any 5 of 40 computers to be rebooted, giving more than 650K ground actions. Both domains have a high amount of stochasticity and strong immediate reward signals meaning that only shallow planning horizons are required. Thus, we use a horizon of h=2 for ILP-HOP, noting that larger horizons did yield improved performance.

For the smallest two problems, 2 and 3, in both domains PROST is slightly better or similar to ILP-HOP. As the number of actions grows larger for the mid-range problems, 5 and 6, ILP-HOP is slightly better or similar to PROST. For the largest problems, 8 and 10, where the number of actions become very large we see that ILP-HOP shows a significant advantage. This shows that PROST has difficulty scaling in these domains as the number of ground actions grows due to the huge increase in branching factor. It is worth noting from Table 1 that for all problems, on average, ILP-HOP is only using a fraction of allowed 180 seconds per step. For the largest instances we see that the ILPs are almost always solved optimally in approximately 5 seconds for Sysadmin and 20 seconds for Game of Life.

Traffic and Elevators. Problems from these domains have factored actions, but a relatively small number of total ground actions compared to the above domains due to constraints on the action variables. Thus, they are better suited to tree search approaches compared to the previous two domains. For Traffic we set the planning horizon for ILP-HOP to be the number of cells between intersections + 4, which is sufficient for seeing the consequences of interactions between intersections. For Elevators we set the horizon to be the number of floors in the buildings + 1, which is large enough to allow any passenger to reach the target floor.

For the small and medium sized problems, 2, 3, 6, and 8, PROST tends to outperform ILP-HOP by a small margin on Traffic and larger margin on Elevators. For the larger problems 8 and 10 ILP-HOP is comparable to PROST, and is significantly better for Traffic 10. One observation from Table 1 is that the ILPs generated for the Elevators domain appear to be more difficult for CPLEX to solve optimally with the percentage of ILPs being optimally solved dropping to as low as 21% for problem 8. The average ILP solutions times are also significantly larger for Elevators even for smaller problems. This is perhaps one reason for the mediocre performance in this domain. Rather, for Traffic 10, where ILP-HOP is doing well, 98% of ILPs were optimally solved.

4.4 Varying the Number of Futures

Figure 1 shows the performance of ILP-HOP with 180 seconds per step for different numbers of futures (1, 3, 5, 7, 10) on one of the more difficult problems from each domain. The number of futures controls a basic time/accuracy tradeoff. The size of the ILPs grows linearly with the number of futures, which typically translates into longer solutions times. However, with more futures we can get a better approximation of the true HOP policy with less variance, which can be advantageous in domains that are well suited to HOP.

The Sysadmin domain is an example where increasing the number of futures leads to steadily improving performance and even appears to have a positive slope at 10 futures. For the other domains, however, there is little improvement, if any, after reaching 3 futures. For Copycat and Game of Life the percentage of optimally solved ILPs was close to 100% for all numbers of futures. Thus, the lack of significant performance improvement may be due to having already reached a critical sampling threshold at 3 futures, or perhaps significantly more futures than 10 are required to see a significant improvement.

Traffic gives an illustration of the potential trade-off involved in increasing the number of futures. We see performance improve when going from 1 to 5 futures. However, the performance then drops past 5 futures. A possible reason for this is that a smaller percentage of the ILPs are solved optimally within the 180 second time limit. In particular, the percentage of ILPs solved optimally dropped from 98% to 59% when the number of futures was increased from 5 to 10. Similarly we see an apparent drop in performance in Elevators for 6, 8, and 10 futures. Here the percentages of optimally solved ILPs drops from 30% to 15% to 1% respectively. These results provide some evidence that an important factor for good performance of ILP-HOP is that a good percentage of the ILPs should be solved optimally.

4.5 Reducing Time Per Step

The above results suggest the importance of using a large enough time per step to allow a large percentage of ILPs to be solved optimally. Here we investigate that observation further by decreasing the time per step for ILP-HOP to 1 second (see Table 1). First, recall that with the exception of Elevators, nearly all ILPs were solved optimally with 180 seconds per step. Rather, for 1 second per step, there are many problems where the percentage drops significantly. The main observation is that when we see a large drop in the percentage of problems solved when going from 180 seconds to 1 second, there is usually a corresponding drop in average reward. As one example, for Game of Life problem 10, the percentage drops from 100% to 38% and the reward also drops from 723 to 635. Traffic and Elevators are the most extreme examples, where for larger problems the percentages drop to zero or near zero. This is accompanied by very large decreases in reward compared to the performance at 180 seconds. Overall, these results confirm that setting the time per step to allow for ILPs to be solved optimally is an important consideration for ILP-HOP.

4.6 Results Summary

ILP-HOP has shown advantages over PROST in 3 domains (Copycat, Sysadmin and Game of Life), especially for large numbers of ground actions, and was comparable or slightly worse in Traffic and Elevators. This shows that the generated ILPs were within the capabilities of the state-of-the-art ILP solver CPLEX. We also saw that it is important to select a time-per-step that allows the solver to return a high percentage of optimal ILP solutions. For most of our domains, 180 seconds was sufficient to this and quite generous for many problems. Finally, for domains like Navigation we observed that HOP does not perform well due to the inherent optimism in the HOP policy.

5 Summary and Future Work

We presented a new approach to probabilistic planning with large factored action spaces via reduction to integer linear programming. The reduction is done so as to return the same action as previous HOP algorithms but without needing to enumerate actions, allowing it to scale to factored actions spaces. Our experiments demonstrated that the approach is feasible in that ILPs for challenging benchmark problems can be formulated and solved, and that ILP-HOP can provide competitive performance and in some cases superior performance over the state-of-the-art. The experiments also showed that the algorithm can tolerate some proportion of ILPs that are not solved optimally, but that planning performance does depend on this factor. This raises several interesting questions for future work. After the initial introduction of planning as satisfiability, significant gains in performance were enabled by careful design of problem encodings. This was possible because major strategies of SAT solvers were easy to analyze but such an analysis is more challenging for ILP solvers. Another important question is whether one can extend this approach to avoid the HOP approximation and optimize the original probabilistic planning problem. This, in fact, can be captured via ILP constraints, but the resulting ILP is likely to be much more complex and encounter more significant run time challenges. An alternative, as in (Kumar, Wu, and Zilberstein 2012; Xue, Fern, and Sheldon 2014), would be to use dual decomposition to simplify the resulting ILP. We hope to explore this approach in future work.

Acknowledgments

This work is partially supported by NSF under grants IIS-0964457 and IIS-0964705. We are grateful to the authors of the RDDL package, and the Prost planner for making their code publicly available. We are grateful to Aswin Raghavan for providing us with his RDDL to tree translator.

References

Bonet, B., and Geffner, H. 2012. Action selection for MDPs: Anytime AO* versus UCT. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of AI Research (JAIR)* 11:1–94.

Chang, H. S.; Givan, R. L.; and Chong, E. K. 2000. On-line scheduling via sampling. In *Proceedings of the Conference on Artificial Intelligence Planning and Scheduling*.

Chong, E. K.; Givan, R. L.; and Chang, H. S. 2000. A framework for simulation-based network control via hindsight optimization. In *IEEE CDC '00: IEEE Conference on Decision and Control*.

Domshlak, C., and Hoffmann, J. 2006. Fast probabilistic planning through weighted model counting. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-quality policies for the canadian traveler's problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Feng, Z., and Hansen, E. A. 2002. Symbolic heuristic search for factored markov decision processes. In *Proceedings of the National Conference on Artificial Intelligence*.

Feng, Z.; Hansen, E. A.; and Zilberstein, S. 2003. Symbolic generalization for on-line planning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Hubbe, A.; Ruml, W.; Yoon, S.; Benton, J.; and Do, M. B. 2012. On-line anticipatory planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Hyafil, N., and Bacchus, F. 2004. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of the European Conference on Artificial Intelligence*.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the European Conference on Artificial Intelligence*.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Kleywegt, A. J.; Shapiro, A.; and Homem-de Mello, T. 2002. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization* 12(2):479–502.

Kolobov, A.; Dai, P.; Mausam; and Weld, D. 2012. Reverse iterative deepening for finite-horizon mdps with large branching factors. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Kumar, A.; Wu, X.; and Zilberstein, S. 2012. Lagrangian relaxation techniques for scalable spatial conservation planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Majercik, S. M., and Littman, M. L. 1998. Maxplan: A new approach to probabilistic planning. In *Proceedings of the Conference on Artificial Intelligence Planning and Scheduling*.

Majercik, S. M., and Littman, M. L. 2003. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence* 147(1):119–162.

Ng, A. Y., and Jordan, M. 2000. Pegasus: A policy search method for large MDPs and POMDPs. In *Proceedings of the Conference on Uncertainty in artificial intelligence*.

Puterman, M. L. 2005. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Raghavan, A.; Joshi, S.; Fern, A.; Tadepalli, P.; and Khardon, R. 2012. Planning in factored action spaces with symbolic dynamic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Raghavan, A.; Khardon, R.; Fern, A.; and Tadepalli, P. 2013. Symbolic opportunistic policy iteration for factored-action MDPs. In *Advances in Neural Information Processing Systems*.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): language description.

Wu, G.; Chong, E.; and Givan, R. 2002. Burst-level congestion control using hindsight optimization. *IEEE Transactions on Automatic Control* 47:979–991.

Xue, S.; Fern, A.; and Sheldon, D. 2014. Dynamic resource allocation for optimizing population diffusion. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Yoon, S.; Fern, A.; Givan, R. L.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Yoon, S.; Ruml, W.; Benton, J.; and Do, M. B. 2010. Improving determinization in hindsight for online probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*.