FIRST ORDER DECISION DIAGRAMS FOR DECISION THEORETIC PLANNING

A Dissertation
submitted by
Saket Joshi

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in
COMPUTER SCIENCE
TUFTS UNIVERSITY
August 2010

ADVISER: Professor Roni Khardon

Abstract

Compact representations of complex knowledge form the core of solutions to many problems in Artificial Intelligence. Sequential decision making under uncertainty is one such important problem and Decision Theoretic Planning (DTP) has been one of the most successful frameworks for this task. Recent advances in DTP have focused on generating efficient solutions for Relational Markov Decision Processes (RMDP), a formulation that models problems that are naturally described using objects and relations among them. The core contribution of this thesis is the introduction of compact representation schemes for functions over relational structures, and associated algorithms that together lead to efficient solutions of RMDPs. Our First Order Decision Diagrams (FODD) representation captures an expressive class of functions generalizing existential quantification in logic to real valued functions, and the Generalized FODDs (GFODDs) capture both existential and universal quantification. The thesis develops several algorithms for composition and logical simplification of functions represented by FODDs and GFODDs using theorem proving and model-checking methods. We prove various theoretical properties on their correctness and their applicability in the context of solutions for RMDPs. Through implementation, experimentation and empirical evidence we demonstrate the success of FODD-based algorithms to solve RMDPs, by applying them to solve stochastic planning problems that have been used as challenge benchmarks in planning research.

Acknowledgements

It has been my good fortune to have Roni Khardon as my Ph.D. adviser. Working with him has been a great learning experience for me. His expert guidance not only made this work possible but also shaped my thinking process and my approach towards research problems. While continually striving to match his abilities, I will miss his gifted intellect, his unique talent in formalizing ideas, his patience, and his encouragement in my future endeavors.

I am very grateful to my committee members, Carla Brodley, Anselm Blumer, Prasad Tadepalli and Eric Miller for the time and effort they spent reviewing my thesis; especially to Carla Brodley for the most effective words of encouragement during a time of crisis and to Prasad Tadepalli for his unconditional help and support throughout my graduate student life.

I owe big thanks to Scott Sanner and Kristian Kersting for their not infrequent mentoring and for invaluable aid in the form of ideas, insights, discussions, suggestions and software.

Finally, my gratitude goes to my family and friends; especially to Avi kaka, without whom I might never have had an interest in science, and to Uday kaka and Manik mavshi whose presence made me feel like I have parents in Boston.

Dedication

To Aai and Baba, whose undying love and sacrifice has made me all that I am today; and to Tai who has been not only the most loving elder sister but also a great source of inspiration.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Many problems in Artificial Intelligence (AI) can benefit from compact representations of complex knowledge. One such problem is that of building agents that interact optimally with an environment in order to achieve a certain objective. More formally, starting with some knowledge about the environment such agents either search, reason or learn to take actions (or build a policy) in the world while optimizing some reward criterion. Various forms of this problem have been studied in the literature. For instance,

- The world could be fully observable, partially observable or unobservable.
- The dynamics of the world could be deterministic, probabilistic or adversarial.
- The objective could be to reach a goal state or to achieve maximum reward.
- The state and action spaces could be discrete or continuous.

Classical planning (e.g., (Fikes & Nilsson, 1971)) addresses one of the simplest versions of this problem where the world is observable, the domain dynamics are deterministic, state and action spaces are discrete and the objective of the agent is to start from a concrete state and reach a concrete goal state. In this thesis we focus on the version where the world is fully observable, the domain dynamics are probabilistic but known in advance, the state and action spaces are discrete and the agents objective is described by a reward function.

This problem is, therefore, that of sequential decision making under uncertainty and is more general than classical planning. Markov Decision Processes (MDP) have become the de-facto standard model for such problems. Some concrete examples of such problems are as follows.

- Planning in Logistics Applications: The agent must maximize goods delivered while minimizing resource consumption.
- Emergency Response Services: A fire response scheduling system must minimize the time and resources required per emergency.
- Robot Navigation: A robot must take the right course of action to get to its destination from the source.

As the examples illustrate, the domains of interest are best described using objects and relations among them. For example, in the logistics application we have packages, vehicles and locations and their corresponding configuration. MDPs respecting such structured state and action spaces are known as Relational MDPs (RMDP). The following are some typical aspects of such real world sequential decision making problems.

- The agent's state space can be very large. For example there could be numerous vehicles to transport numerous parcels between numerous destinations in the logistics application. Each configuration of these objects is a possible state of the world.
- The effect of the agent's actions on the environment can be uncertain. Robots can slip and skid on a smooth floor. Time taken to deliver packages can vary. The returns on investments can be uncertain.
- The agent's objective can be complex. In the logistics world, the objective could be to maximize the total number of packages delivered while minimizing time and resources. The robot's objective could be to minimize average energy consumption per route.

Research on such problems has progressed by solving simpler versions that are generated by restricting the setting and abstracting away certain parts of the problem. As an example we will explain a simplified logistics domain in some detail here. Following Veloso (1992) and Boutilier, Reiter, and Price (2001), this toy domain has been used by several authors (Kersting, van Otterlo, & De Raedt, 2004; Sanner & Boutilier, 2009; Wang, Joshi, & Khardon, 2008) to demonstrate ideas and methods. This domain will also be used as a running example to explain many ideas in this thesis.

1.1 The Logistics Domain

In the logistics domain the world consists of three types of objects, namely boxes, trucks and cities. A box can be either in a city or on a truck. The agent's objective is to transport boxes from their source cities to their destination cities. The trucks can be used for transportation. A world state is described by specifying the location of every box (in a city or on a truck), the location of every truck, and whether it is raining or not. There are three actions available to the agent. At any time step, the agent can either load a box from a city onto a truck, unload a box from a truck into a city, or drive a truck from one city to another. In this simplified domain, every city is reachable from every other city in one time step. A truck can carry any number of boxes. Common sense domain constraints apply, e.g., a box or truck cannot be in two places at once.

The drive action is deterministic. The load and unload actions are probabilistic and in this simplified domain they either succeed leading to their expected outcomes or fail where nothing in the world changes. The load action succeeds with probability 0.99 and fails with probability 0.01. The probabilities of success and failure of the unload action are conditioned on whether it is raining or not. If it is raining, unload succeeds with probability 0.7 and fails with probability 0.3. If it is not raining, unload succeeds with probability 0.9 and fails with probability 0.1.

1.2 Our Approach

Notice that even in such abstracted worlds, the typical aspects of real world problems listed above are preserved. Therefore solution algorithms for such problems must handle large state spaces with uncertain action effects and rich representations for complex objectives. The problem of solving MDPs has been a widely studied in Operations Research and Computer Science. Most solution methods, however, represent the state space either as a flat set of monolithic world states or as defined by values assigned to state feature variables. Although there are dynamic programming algorithms that solve the MDP in time polynomial in the size of the state space, the size of the state space itself is exponential in the number of state feature variables. Hence even for small problems, the MDP can be prohibitively expensive to solve by state space enumeration. This is known as the curse of dimensionality (Bellman, 1957). Recent work has addressed this problem by taking advantage of internal structure in the problem definition (Boutilier, Dearden, & Goldszmidt, 1995, 1999; Hoey, St-Aubin, Hu, & Boutilier, 1999). However even these algorithms cannot handle very large problem instances. In addition, a solution for every problem instance (e.g., one with five boxes and one with ten boxes) has to be generated separately and these algorithms cannot take advantage of similarities in problem instances to use the solution of one problem in solving another. All is not lost, however, and there is potential in the observation that for domains like logistics, the world is naturally represented by objects and relations among them rather than as a set of state feature variables. Thus there exists rich relational structure in these problems that can be exploited to further counter the curse of dimensionality.

In this thesis we follow the approach of Boutilier et al. (2001), who developed the Symbolic Dynamic Programming (SDP) algorithm to solve problems of sequential decision making under uncertainty. The main idea in SDP is to abstract the relational structure of the underlying domain and generate a solution in terms of the relational structure rather than actual domain objects. Such a solution is independent of the actual problem instance and is valid for all domain sizes. Given an abstract form of the objective, such a solution has to generated only once. For example, in the logistics world our objective could be to transport at least one box to Paris. Once SDP generates a solution for this problem, the same solution is valid for all problems with the same objective independent of the number of boxes, cities and trucks in the domain. Our main contribution is the introduction and use of a new compact knowledge representation to capture complex objectives and domain dynamics under the SDP algorithm. We develop algorithmic tools for this knowledge representation and make a case for the applicability of this approach through theoretical results and empirical evidence. In the next section we present our contributions in more detail.

1.3 Major Contributions

Following are the main contributions of this work. Most of this work has been published in conference and journal papers (Wang, Joshi, & Khardon, 2007; Wang et al., 2008; Joshi & Khardon, 2008; Joshi, Kersting, & Khardon, 2009, 2010).

1. **First Order Decision Diagrams:** A First Order Decision Diagram (FODD) is a compact knowledge representation for real valued functions over relational structures. That is, FODDs map every possible world state to a real value. Such functions are very useful in defining utilities and probabilities of world states in an MDP. We modify and extend the approach of Groote and Tveretina (2003) to develop the FODD representation and algorithms to manipulate them. FODDs can be viewed as a relational extension of Algebraic Decision Diagrams (Bryant, 1992; McMillan, 1993; Bahar, Frohm, Gaona, Hachtel, Macii, Pardo, & Somenzi, 1993). When restricted to Boolean valued leaves, FODDs represent function free First Order formulas with existentially quantified variables. We demonstrate the use of FODDs to represent MDPs and develop an SDP algorithm for the FODD representation.

2. **Theorem Proving Reductions:** SDP based algorithms (Boutilier et al., 2001; Kersting et al., 2004; Sanner & Boutilier, 2009) have to perform reasoning in First-Order logic in order to represent and manipulate partitions of the state space. Therefore all such algorithms require the need for logical simplification of relational formulas for practical implementation. Similarly, logical reasoning with FODDs creates redundancies in the diagrams. Special operators are needed to identify and remove these redundancies. Although the removal of redundancies or lack thereof does not affect correctness of the representation, these operators are required for any practical application of FODDs. We present new reduction operators for FODDs based on theorem proving of First Order formulas. The idea is to use logical implication to identify redundant parts of the diagram and remove them.

3. **Model Checking Reductions:** Theorem proving reductions have a few drawbacks. Proving logical implication is an expensive operation. In addition, domain constraints have to be explicitly specified as background knowledge. To mitigate these

issues, we introduce a new paradigm for reduction of FODDs based on model-checking and prove its superiority over theorem proving reductions. We present theoretical and practical versions of model-checking reductions and provide proofs of correctness and completeness.

4. **Weighted Goal Ordering Heuristic:** SDP based algorithms have been motivated by problems in probabilistic planning. Planning problems pose the task of reaching a concrete goal state from a concrete start state in the world. SDP based algorithms, however, are designed to solve the domain at the abstract level and planning for a concrete domain instance will lose the benefits of abstraction. Therefore Sanner and Boutilier (2009) introduced a method to plan for generic goals at the abstract level, and given a concrete instance and goal, use a goal decomposition to put together solutions to sub-problems. However, their solution makes an assumption that the decomposed parts of the goal are independent of each other. This assumption is unrealistic in some domains. We present a new heuristic for goal decomposition that is not dependent on this assumption. We show evidence of its superiority over the heuristic of Sanner and Boutilier (2009) in domains where goal serializability is a crucial factor.

5. **FODD-Planner:** We incorporate all the above ideas for FODD manipulation and present a prolog based software system that implements the SDP algorithm with the FODD representation. We demonstrate this system by solving stochastic planning problems and showing performance comparable to top ranking systems from the International Planning Competitions. This shows that abstraction through compact representation is a promising approach for solving sequential decision making problems. The results make a very good case for the use of FODDs in representing functions over relational structures (which are important for many applications like Statistical Relational Learning) and demonstrate their efficient manipulability.

6. **Self-Taught Planning:** Inspired by the efficiency of model-checking reductions we develop a new paradigm for planning by learning. The idea is to provide FODD-PLANNER with a small "training set" of world states of interest, but no indication of optimal actions in any states. The FODD-PLANNER uses this training set to "focus"

its logical simplification in model-checking reductions to include only formulas relevant for these states. We also show that such training examples can be constructed on the fly from a description of the planning problem. Thus we can bootstrap our planner to get a self-taught planning system. We show drastic improvements in planning efficiency on a variety of IPC domains. Although we employ the FODD-PLANNER to demonstrate the self-taught planning paradigm we believe that this technique is applicable with any SDP based system.

7. **Generalized FODDs:** FODDs are compact and expressive but when considered as logical formulas they are limited to existential quantification. We present Generalized FODDs (GFODD), where we extend the representation power to arbitrary aggregation where aggregation is a generalization of quantification. Every FODD is also a GFODD. GFODDs are thus very expressive structures that share the same compactness advantages as FODDs. We discuss several properties of GFODDs and present reductions for an important subset of GFODDs. We also identify conditions under which GFODDs can be composed or combined by a simple procedure.

8. **VI-GFODD:** GFODDs can represent complex functions over relational structures and can thus be employed instead of FODDs in applications such as sequential decision making to enhance expressive power of representation. We show that the same FODD-based SDP algorithm used above is valid when GFODDs are used as the underlying representation. Finally we prove the correctness of this algorithm, VI-GFODD, for a very expressive subset of GFODDs. Within this representation we can capture objectives like *transport at least one box to Paris and all trucks to London* in the logistics domain. This was not possible with FODDs.

1.4 Thesis Overview

The thesis proceeds as follows. We start by providing technical background in Chapter 2. This includes an overview of the literature on planning under uncertainty and solution techniques for MDPs, factored MDPs and Relational MDPs.

In Chapter 3 we introduce First Order Decision Diagrams (FODD) and discuss many of their properties. We also discuss the process of reasoning with FODDs and present some operators to reduce FODDs. The results in Chapter 3 have previously appeared in the thesis of Wang (2008). Chapter 4 addresses the deficiencies of the set of reduction operators presented in Chapter 3 by introducing new reduction operators for FODDs. In Chapter 5 we demonstrate the first empirical evidence and practical applicability of FODDs by presenting our SDP based MDP solver, FODD-PLANNER, and results of experiments of the application of FODD-PLANNER to stochastic planning problems from the International Planning Competition. FODD-PLANNER implements the FODD manipulation algorithms presented in Chapters 3 and 4.

In Chapter 6 we introduce a new paradigm for reduction of FODDs based on model-checking and prove that it gives much stronger reduction guarantees than the theorem proving reductions of Chapters 3 and 4. We also present practical versions of this reduction that can be easily implemented. Then in Chapter 7 we incorporate the new reductions of Chapter 6 in FODD-PLANNER and develop a new paradigm for planning. We also show empirical evidence of drastic improvements in planning efficiency of FODD-PLANNER by shifting focus to the new reductions.

In Chapter 8 we present Generalized FODDs (GFODDs), discuss their properties (combination and reduction) and prove correctness of a GFODD based SDP algorithm, VI-GFODD.

We conclude in Chapter 9 with a discussion and perspective for future work.

Chapter 2

Background

In this chapter we provide the basic background and the context for the thesis. In the process we also elaborate on related work.

2.1 Planning Under Uncertainty

Planning is the problem of getting from a start state to a state satisfying some goal conditions using actions which move the agent according to known transition dynamics. In classical planning the dynamics are deterministic. Therefore, the problem can be seen as a search for a sequence of actions that achieves the goal. With uncertainty the dynamics are non-deterministic and there are several formalisms capturing planning under uncertainty. In our work action dynamics are probabilistic, a formalism known as stochastic planning. Deterministic Planning is a relatively mature field with a number of planning formalisms and systems developed over the years. The STRIPS planning system (Fikes & Nilsson, 1971) led a generation of automated planning research and produced a number of successful systems for deterministic planning using various paradigms like partial order planning (Penberthy & Weld, 1992), planning based on planning graphs (Blum & Furst, 1997), planning by satisfiability (Kautz & Selman, 1996) and heuristic search (Bonet & Geffner, 2001).

These ideas were later employed in solving the problem of stochastic planning (Blum & Langford, 1998; Weld, Anderson, & Smith, 1998; Majercik & Littman, 2003; Yoon, Fern,

& Givan, 2007; Teichteil-Koenigsbuch, Infantes, & Kuter, 2008). Of these, approaches using forward heuristic search with a heuristic function based on the planning graph (Blum & Furst, 1997) have been very successful at the recent International Planning Competitions (Yoon et al., 2007; Teichteil-Koenigsbuch et al., 2008). A deeper overview of the different approaches to planning is beyond the scope of this thesis. An up to date overview is given by Russel and Norvig (2010).

In the case of planning under uncertainty, straight line plans do not guarantee achievement of the goal due to the stochastic nature of the underlying world. A Markov decision process is a natural formalism for optimizing actions under stochastic dynamics and its application to planning is known as known as Decision Theoretic Planning (DTP) (Boutilier, Dean, & Hanks, 1999). A number of the algorithms and systems for DTP mentioned in this chapter have been motivated by and employed in solving stochastic planning problems.

In recent years, the International Planning Competition (IPC) has been instrumental in the development of efficient algorithms and systems for planning. There have been six competitions so far since the AIPS competition in 1998 (McDermott, 1998). The last three IPCs since ICAPS 2004 promoted a separate track for planning under uncertainty (Littman & Younas, 2004; Gerevini, Bonet, & Givan, 2006; Bryce & Buffet, 2008).

2.2 Markov Decision Processes

A Markov decision process (MDP) is a mathematical model of the interaction between an agent and its environment. The environment can be dynamic and stochastic. The agent's objective is to act optimally (or near optimally) in the environment. Puterman (1994) provides a comprehensive discussion of MDPs. Formally an MDP is a 5-tuple $\langle S, A, T, R, \gamma \rangle$ defining

- A set of fully observable states S.
- A set A of actions available to the agent.
- A state transition function defining the probability $P(s'|s, a)$ of getting to state s' from state s on taking action a.

- A reward function $R(s, a)$ defining the immediate reward received by the agent for being in state s and taking action a. To simplify notation we assume throughout this thesis that the reward is independent of a so that $R(s, a) = R(s)$. However the general case does not lead to significant difficulties.

- A discount factor $0 \leq \gamma \leq 1$.

The reward function provides a mathematical way of encoding the agent's objective in the environment. Informally the agent's goal is to take actions that help accumulate as much reward as possible. For problems that do not have a finite horizon, the total reward achieved can be infinite. Thus to help quantify distinctions between policies either the average per-step reward model or a discounted reward model is considered. In this thesis we use the discounted model which is described next. For episodic tasks such as planning it provides an incentive to find short solutions. Another alternative in this case uses an "absorbing state" when the goal is achieved so that the reward can only be obtained once in any episode. We discuss this model later in the thesis.

In the discounted model, the objective of solving an MDP is to generate a policy (a mapping from states to actions) π^* that maximizes the agent's total, expected, discounted reward. The value (or utility) of a state $V^{\pi}(s)$ under a policy π is the total, expected, discounted reward achieved by the agent starting from s and following π. That is, $V^{\pi}(s) = \sum_0^{\infty} \gamma^i R(s_i) \mid s_0 = s$. Intuitively the expected utility or value of a state under the optimal policy is equal to the reward obtained in the state plus the discounted value of the state reached by the best action in the state. This is captured by the Bellman equation as

$$V^*(s) = Max_a[R(s) + \gamma \Sigma_{s'} P(s' \mid s, a) V^*(s')] \tag{2.1}$$

that is well known to define the optimal value function V^*. For tasks such as planning where the objective is to reach a set of goal states, the value function $V(s)$ encodes the "distance" from the state s to the goal. The optimal value function and the transition function together define the optimal policy. Therefore solving an MDP can be reduced to solving the Bellman equation. This has been a very productive approach over several decades. Next we will look at some standard algorithms for solving MDPs.

2.2.1 Value Iteration

Value Iteration (VI) (Bellman, 1957) is a dynamic programming algorithm that treats the Bellman equation as an update rule. Starting from an arbitrary value function $V^0(s)$, the value of every state is iteratively updated until convergence using the rule

$$V^t(s) \leftarrow Max_a[R(s) + \gamma \Sigma_{s'} P(s' \mid s, a) V^{t-1}(s')]. \tag{2.2}$$

For practical solutions, convergence is usually defined as ϵ-optimality meaning that convergence is obtained when the difference between the current and the optimal value function is less than ϵ i.e. $\mid V^{t+1}(s) - V^*(s) \mid \leq \epsilon$ (Puterman, 1994). Algorithmically the test $\mid V^{t+1}(s) - V^t(s) \mid \leq \frac{\epsilon(1-\gamma)}{2\gamma}$ guarantees ϵ-optimality. Once the optimal value function is known, a policy can be generated by assigning to each state the action that maximizes expected value. VI, therefore, ignores policies altogether until the ϵ-optimal value function is discovered.

The update is generally performed in two steps. The back-up or regression step calculates the Q-function for every state-action pair.

$$Q^t(s, a) \leftarrow R(s) + \gamma \Sigma_{s'} P(s' \mid a, s) V^{t-1}(s') \tag{2.3}$$

The maximization step calculates the t-step-to-go value function.

$$V^t(s) \leftarrow Max_a Q^t(s, a) \tag{2.4}$$

Figure 2.1(b) shows an illustration of dynamic programming methods for calculating the value function. The current value function estimate ($V^i(s)$ depicted by the squares) is backed up over actions using the Bellman update to give the next value function estimate ($V^{i+1}(s)$).

2.2.2 Policy Iteration

Policy iteration (PI) (Howard, 1960) is another dynamic programming solution, but unlike VI it starts with an arbitrary policy π and “improves” it until convergence to the optimal policy. Given an initial policy π^0, the algorithm iterates over two steps:

1. Policy Evaluation: Calculate the value V^{π^i}, of policy π^i. This is done by solving the Bellman equation for the fixed policy π^i.

$$V^{\pi^i}(s) = R(s) + \gamma \Sigma_{s'} P(s' \mid s, \pi^i(s)) V^{\pi^i}(s') \tag{2.5}$$

2. Policy Improvement: Find a new policy π^{i+1} that is greedy with respect to V^{π^i}

Convergence is reached when $\pi^{i+1} = \pi^i$. One advantage of policy iteration is that the policy evaluation step consists of solving simultaneous linear equations. Hence for moderate size state spaces one can utilize off-the-shelf linear algebra solvers to perform this step.

2.2.3 Modified Policy Iteration

PI calculates V^{π^i} exactly before improving the policy. For policy improvement, however, a close estimate of V^{π^i} is not necessary. The idea behind Modified Policy Iteration (MPI) (Puterman & Shin, 1978) is to generate a rough estimate of V^{π^i} and run the policy improvement step before V^{π^i} converges. This is done by a sequence of "policy restricted" value backups using the Bellman update but fixing the actions according to π^i. MPI can thus be viewed as covering the entire space between PI at one end and VI at the other (every update of VI implicitly defines a policy). While PI takes bigger steps through policy space towards the optimal policy than MPI, each step of MPI is cheaper to calculate. Overall, MPI is often more efficient than either VI or PI when suitably optimized.

2.2.4 Linear Programming

The problem of solving for the value function can be cast as a linear programming problem (Puterman, 1994) in the following way

Variables: $V(s)\ \forall s \in S$
Minimize: $\Sigma_{s \in S} V(s)$
Subject to: $V(s) \geq R(s) + \gamma \Sigma_{s' \in S} P(s' \mid s, a) V(s')\ \forall s \in S, a \in A$

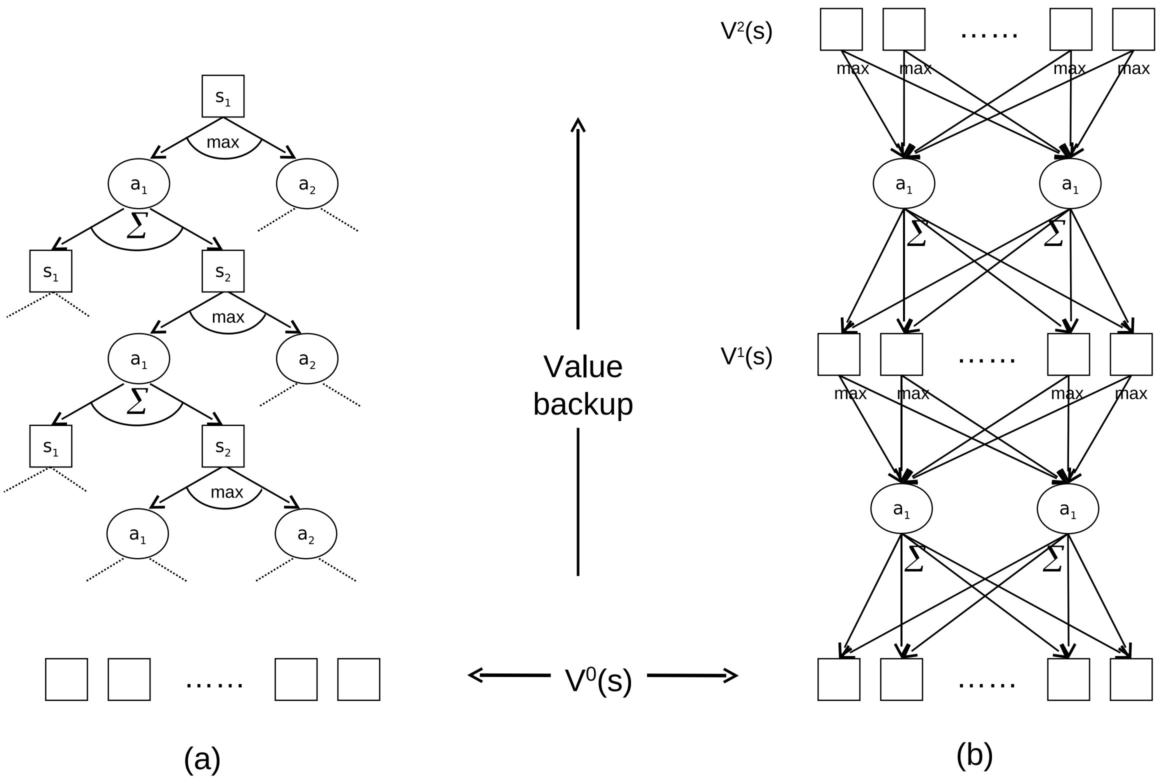


Figure 2.1: Propagation of value in the MDP by (a) forward search and (b) dynamic programming

The constraints impose a lower bound on the value function. The intuition is that the total, expected discounted reward obtained by starting in state s and following the optimal policy is at least as much as $R(s)$ plus the total, expected, discounted reward obtained by taking some action in s and following the optimal policy from there on. The linear program, thus finds the smallest value assignment that is at least as large as the total, expected, discounted reward it guarantees.

2.2.5 Search

$V^*(s)$ can also be calculated by forward search from state s up to a prespecified depth. Forward search with depth equal to the MDP horizon calculates $V^*(s)$ exactly. If the depth is smaller than the MDP horizon, the error of the value function can be bounded. This method, therefore always produces an approximate value function for infinite horizon

MDPs. As shown in Figure 2.1(a), the search tree is an AND-OR tree consisting of alternate levels of states and actions. The value is propagated upwards from the leaves, summing over the state levels and maximizing over the action levels. Forward search, however, requires exponential time in the depth to calculate the value of every state. Hence practical search based solvers employ some heuristic to prune the search space. AO* (Nilsson, 1971; Martelli & Montanari, 1973) is such a heuristic search algorithm. However, AO* enters an infinite search expansion loop when the AND-OR graph is cyclic, which is often the case with MDPs. In order to handle this case Hansen and Zilberstein (2002) combined AO* search with dynamic programming and introduced LAO* - an AO* that can handle loops in the AND-OR graph. Since value updates are performed by dynamic programming, the presence of loops in the AND-OR graph does not preclude convergence of the policy. Unlike VI and PI, however dynamic programming is performed only on the states along the best solution path to recently expanded nodes in the AND-OR graph.

2.2.6 Asynchronous Value Updates

The Bellman update does not have to be applied to all states at once. By choosing a useful subset of the state space to apply the updates one can hope for faster convergence. There have been some important advances based on this idea.

Real Time Dynamic Programming (RTDP) (Barto, Bradtke, & Singh, 1995), is an extension of the search algorithm Learning Real Time A* (LRTA) (Krof, 1990) to stochastic problems. RTDP solves the MDP by combining dynamic programming and search. Starting with an admissible value function, RTDP runs episodes or trials simulating the MDP. Each trial starts with a randomly selected state from a set of initial states and proceeds by choosing actions according a policy that is greedy with respect to the current value function. The episode ends when the goal is reached or the step limit is exceeded. The value function is updated by executing the Bellman backup for all states encountered in the episode. When the initial value function is admissible, RTDP eventually converges to the optimal value function (Barto et al., 1995). The advantage of RTDP is apparent when only few states are reachable from the initial state as the algorithm does not waste resources updating values of other states.

Prioritized Sweeping (Moore & Atkeson, 1993) is based on the intuition that states with the largest value update have the most effect on the value function and hence should be given priority during a round of Bellman backups. In this algorithm an explicit priority queue is maintained according to which states are scheduled for backup. The priority of a state in the queue is in direct proportion to the amount of value change incurred in the backup of the state's successors.

Another approach to simulation based planning is the UCT algorithm (Kocsis & Szepesvari, 2006). UCT is a Monte-Carlo value estimation technique where action selection in the simulated trials is dependent on the upper confidence bound of the action value. UCT has shown success in planning to play the game of GO (Gelly & Wang, 2006; Gelly & Silver, 2007) and real time strategy games (Balla & Fern, 2009).

2.2.7 Reinforcement Learning

The MDP solutions we have looked at so far assume that the full description of the MDP is available to the solver. However, in many real world tasks (e.g. robot soccer) it is difficult for an expert to specify the reward function or the transition function precisely. Thus the agent interacts with the environment and must learn to act from the observed transitions and rewards. This discipline is known as Reinforcement Learning (RL) (Sutton & Barto, 1998). There is a vast literature on RL which is a very active area of research and it is beyond the scope of this thesis to give a deeper overview. An excellent introduction is given by Sutton and Barto (1998). However, when the agent does have a model of the world, either learned or given, the planning problem, that is identifying an optimal policy is the same as the one studied in this thesis.

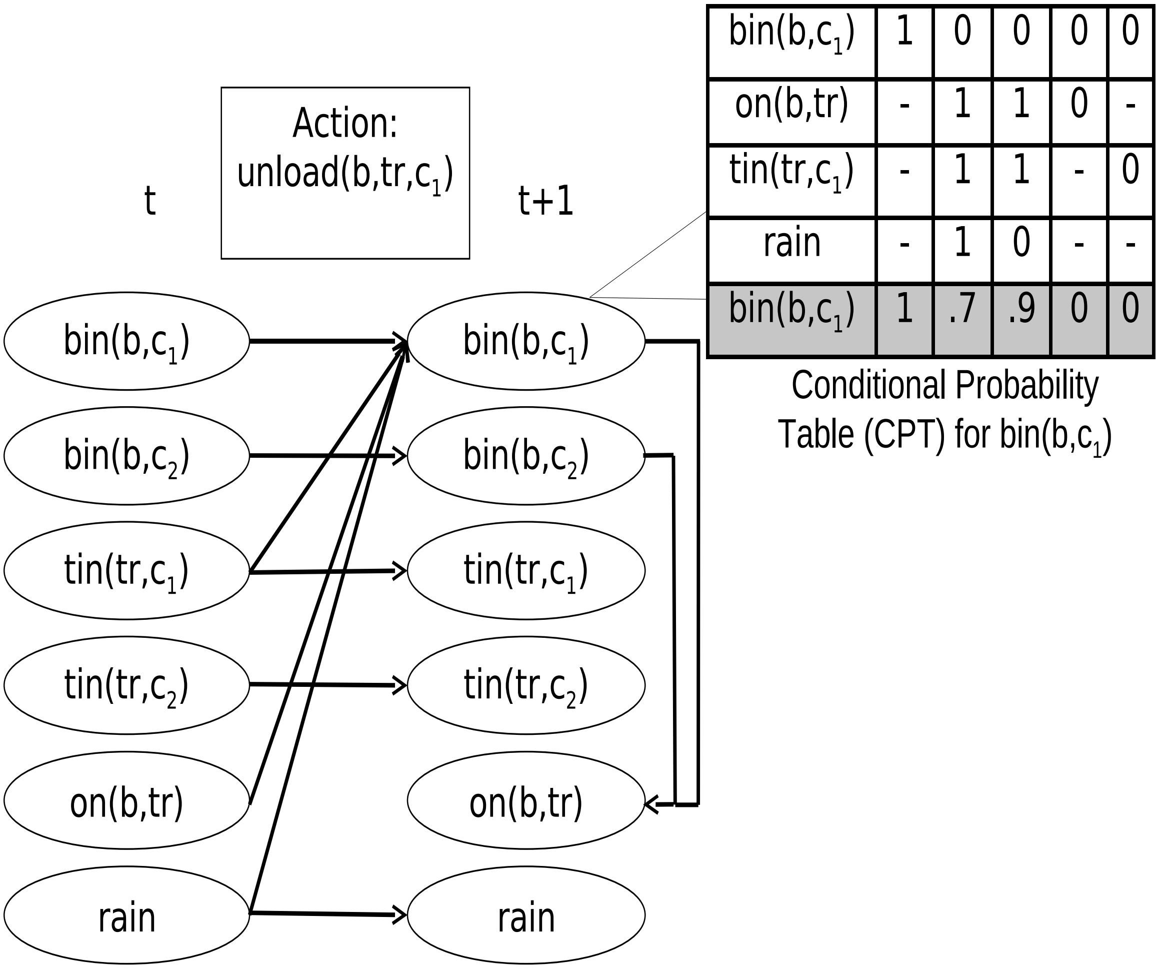
2.3 Factored MDPs

Early solutions to MDPs required enumeration of the state space. Although VI and PI run in time polynomial in $|S|$ and $|A|$, owing to the curse of dimensionality (Bellman, 1957), even for reasonably small problems, the state space can be very large. This can be seen easily for propositionally factored domains when the state is defined by N binary variables and the number of possible states is 2^N. A logistics problem with as few as ten cities, ten boxes and five trucks would be prohibitively expensive to solve by state space enumeration. Factored MDP solvers address this problem by taking advantage of structure in the domain. One source of domain structure is the representation of states using state feature variables. Structure in the domain can be implicitly leveraged by representing state values as functions of state features rather than states themselves.

Domain structure can also be expressed explicitly through the transition function. Dynamic Bayesian Networks (DBN) (Dean & Kanazawa, 1990) and probabilistic STRIPS (Hanks & McDermott, 1993; Kushmerick, Hanks, & Weld, 1995; Boutilier, Dean, & Hanks, 1996) are useful representations for expressing the transition function in factored MDPs. Although both representations have their strengths (DBNs are better than probabilistic STRIPS when actions have non-correlated effects and vice-versa (Boutilier et al., 1999)) Littman (1997) showed the two to be representationally equivalent in the sense that every STRIPS representation can be converted to a polynomially larger DBN representation.

Figure 2.2 shows an example of a DBN representation for the action $unload(b, tr, c_1)$ from the logistics domain with 1 box, 1 truck and 2 cities. There is a 2-time step DBN fragment for every action in the domain. The network describes how the state (probabilistically) changes from time slice t to time slice $t + 1$. Conditional probability tables (CPT) are associated with every variable in time slice $t + 1$. The CPT associated with the variable $bin(b, c_1)$ in time slice $t+1$ is shown in the diagram. The DBN structure allows us to model dependencies while reducing the number of parameters required. The CPTs are much more memory efficient than the flat transition matrix for this problem which enumerates the state space. Also notice the presence of synchronic arcs (dependencies among variables in the same time slice). These are typical when action effects are correlated.

The probabilistic dependence of the variables from time slice $t+1$ on the variables from time slice t is given by the CPTs. For instance in the CPT for $bin(b, c_1)$ shown, each row of the CPT corresponds to a variable from time slice t except for the last (highlighted) row which refers to the variable $bin(b, c_1)$ from time slice $t + 1$. Each column depicts the truth values of the variables in time slice t and the entry in the final row defines the probability with with $bin(b, c_1)$ is true in time slice $t + 1$ given the truth values of the variables in the



bin(b,c$_1$)	1	0	0	0	0
on(b,tr)	-	1	1	0	-
tin(tr,c$_1$)	-	1	1	-	0
rain	-	1	0	-	-
bin(b,c$_1$)	1	.7	.9	0	0

Figure 2.2: Dynamic Bayesian Networks for Factored MDPs. The figure shows the transition function relative to the action $unload(b, tr, c_1)$ in the logistics domain.

column. For example, the first column shows that if $bin(b, c_1)$ is true in time slice t, then $bin(b, c_1)$ is true in time slice $t + 1$ irrespective of the truth values of other variables. The last column shows that if $bin(b, c_1)$, and $tin(tr, c_1)$ are false in time slice t, then irrespective of the truth value of $on(b, tr)$ and $rain$, $bin(b, c_1)$ is true in time slice $t + 1$ with 0 probability (always false). The predicate $on(b, tr)$ is true at time $t + 1$ only when the box is in neither of the cities. This dependency has been shown by the synchronic arcs. The truth value of other variables can only persist under the action $unload(b, tr, c_1)$. Synchronic arcs cannot be handled by some factored MDP solvers. One could use joint variables to replace variables connected by synchronic arcs. This would cause a blow up exponential in the number of variables joined. But by doing so we could convert a DBN with synchronic arcs to one without synchronic arcs. The equivalent representation in probabilistic STRIPS is as follows:

Action: unload$(box, truck, city)$:
Preconditions: on$(box, truck)$, tin$(truck, city)$
Outcome 1**:** [*Probability* $rain \rightarrow 0.7$, $\neg rain \rightarrow 0.9$] bin$(box, city)$, $\neg$on$(box, truck)$
Outcome 2**:** [*Probability* $rain \rightarrow 0.3$, $\neg rain \rightarrow 0.1$] nothing changes.

Unfortunately the factoring of the transition function itself does not guarantee a compact value function. This is because the state variables interact through parent-child relationships in the DBN. Therefore the n-step-to-go value function that captures interactions over n actions could depend on the combinations of all values of all variables, thus eliminating the advantage of factoring.

An approach that solves factored MDPs effectively and exactly is based on the observation that in the CPTs, the values of some variables could make other variables unimportant. This phenomenon, known as context specific independence (Boutilier, Friedman, Goldszmidt, & Koller, 1996), enables the use of compact data structures like decision trees and Algebraic Decision Diagrams (Bahar et al., 1993) to represent the CPTs. Boutilier et al. (1995, 1999) showed that using decision trees to represent the CPTs, and the reward function, optimal policies and value functions can be generated by a structured version of VI and PI. Dearden (2001) presented a structured prioritized sweeping algorithm based on this idea. Although decision trees leverage context specific independence they could be exponentially larger than ADDs. Noticing this Hoey et al. (1999) replaced the decision trees by ADDs in their decision theoretic planning system SPUDD. SPUDD showed a gain up to 30 times in memory efficiency over the decision tree representation. Extensions of this work using approximations in the ADDs (St-Aubin, Hoey, & Boutilier, 2000) displayed further improvement in efficiency. Following this work, ADDs have been used in several algorithms to represent and solve MDPs. Feng and Hansen (2002) combined the SPUDD approach with forward search to extend LAO* to factored MDP. The SPUDD variant of

Teichteil-Koenigsbuch and Fabiani (2006) participated in the International Planning Competition. Sanner, Uther, and Delgado (2010) showed further gains in efficiency by shifting focus from ADDs to Affine ADDs (Sanner & McAllester, 2005).

Another effective approach has been to combine dynamic programming with function approximation. The value function is described using a parametric representation such as a neural network (Tesauro, 1992) or a linear function of a basis derived from state feature variables (Schweitzer & Seidmann, 1985; Tsitsiklis & Van Roy, 1996). Value function approximations were also used in solutions not based on dynamic programming. One such approach to generating a compact solution restricts the form of the value function. Once again linear value functions are attractive because of their mathematical properties (Koller & Parr, 1999, 2000; Schuurmans & Patrascu, 2001; Guestrin, Koller, Parr, & Venkataraman, 2003b). Since the value function might not be representable by a linear combination of basis functions derived from state feature variables, these approaches are necessarily approximate. But their advantage is that they are not dependent on context specific independence to generate a compact solution.

2.4 Relational MDPs

Propositionally factored representations show an impressive speedup by taking advantage of the propositional domain structure. However, they do not benefit from the structure that exists with objects and relations. A Relational MDP (RMDP) is an MDP where the world is represented by objects and relations among them. A RMDP is specified by

1. A set of world predicates. Each atom, formed by instantiating a predicate using objects from the domain, can be either `true` or `false` in a given state. For example in the logistics domain, world atoms are bin$(box, city)$ (box is in $city$), on$(box, truck)$ (box is on $truck$), and tin$(truck, city)$ ($truck$ is in $city$).

2. A set of action predicates. Each action atom formed by instantiating an action predicate using objects from the domain defines a concrete action. For example in the logistics domain, action atoms are of the form load$(box, truck, city)$ (load

box on to $truck$ in $city$), unload$(box, truck, city)$ (unload box from $truck$ in $city$), drive$(truck, source.city, dest.city)$ (drive $truck$ from $source.city$ to $dest.city$), etc.

3. A state transition function that provides an abstract description of the probabilistic move from one state to another. For example, using probabilistic STRIPS notation, the transition defined by the action load can be described as

   Action: load$(box, truck, city)$
   Preconditions: bin$(box, city)$, tin$(truck, city)$
   Outcome 1**:** [*Probability* 0.99] on$(box, truck)$, $\neg$bin$(box, city)$
   Outcome 2**:** [*Probability* 0.01] nothing changes.

   If the preconditions of the action bin$(box, city)$, tin$(truck, city)$ are satisfied, then with probability 0.99, the action will succeed generating the effect on$(box,\ truck)$, $\neg$ bin$(box,\ city)$. The state remains unchanged with probability 0.01. The effects of actions in RMDPs are usually correlated and cannot be considered to occur independent of one another. Therefore Probabilistic STRIPS is a better representation for the transition function.

4. An abstract reward function describing conditions under which rewards are obtained. For example in the logistics domain, the reward function can be $\exists x$, bin$(x, paris)$ constructed so as to capture the goal of transporting at least one box from its source city to Paris.

Work on RMDPs has been largely motivated by and applied to the problem of stochastic planning. In fact, the languages that have been popular in defining planning domains, e.g. STRIPS (Fikes & Nilsson, 1971), ADL (Pednault, 1989), PDDL (Ghallab, Howe, Knoblock, McDermott, Ram, Veloso, Weld, & Wilkins, 1998; Fox & Long, 2003; Younes, Littman, Weissman, & Asmuth, 2005) have all exploited relational structure of the planning domains. Hence, in a way, it is more natural to think of solving probabilistic planning problems using RMDP solvers. However, RMDP solvers solve entire classes of problems given the planning domain. This might generate an unnecessary overhead when solving simple problems but can be a huge advantage when the domain is very large. This thesis presents one such RMDP solver that has found success in solving probabilistic planning problems.

Boutilier et al. (2001) developed the first VI method for solving RMDPs and provided the Symbolic Dynamic Programming (SDP) algorithm in the context of the situation calculus. This algorithm provided a framework for dynamic programming solutions to RMDPs that was later employed in several formalisms and systems (Kersting et al., 2004; Hölldobler & Skvortsova, 2004; Sanner, 2008; Sanner & Boutilier, 2009). The main advantage of SDP is that state and action predicates are not grounded or enumerated and the MDP is solved to the extent possible at the abstract level making distinctions among states only when the value function requires it. One of the important ideas in SDP was to represent stochastic actions as a finite set of deterministic alternatives under nature's control. This helps separate regression over deterministic action alternatives from the probabilities of action effects. This separation is necessary when transition functions are represented as relational schemas abstracting over the structure of the states. Recall the Bellman update step of the VI algorithm.

$$V^t(s) \leftarrow Max_a[R(s) + \gamma \Sigma_{s'} P(s' \mid s, a) V^{t-1}(s')] \tag{2.6}$$

The SDP algorithm implements this for all states simultaneously at the abstract level. Intuitively, each V^t partitions the state space into "abstract states" where all states in an abstract state have the same value and are thus "equivalent". The basic outline of the relational value iteration algorithm is as follows:

1. **Regression:** The t-step-to-go value function V^t is regressed over every deterministic variant $A_j(\vec{x})$ of every action $A(\vec{x})$ to produce $Regr(V^t, A(\vec{x}))$. At the first iteration V^0 is assigned the reward function. This is not necessary for correctness of the algorithm but is a convenient starting point for VI. $Regr(V^t, A(\vec{x}))$ describes the conditions under which the action alternative $A_j(\vec{x})$ causes the state to transition to some abstract state description in V^t.

2. **Add Action Variants:** The Q-function for each action $A(\vec{x})$ is generated, $Q_{V^t}^{A(\vec{x})} = R \oplus [\gamma \otimes \oplus_j (prob(A_j(\vec{x})) \otimes Regr(V^t, A_j(\vec{x})))]$. In this step the different alternatives of an action are combined. Each alternative $A_j(\vec{x})$ produces a $Regr(V^t, A(\vec{x}))$ from the regression step. All the $Regr(V^t, A(\vec{x}))$s are added, and each is weighted by the

probability of $A_j(\vec{x})$. This produces the parametrized function $Q_{V^t}^{A(\vec{x})}$ which describes the utility of being in a state and taking a concrete action $A(\vec{x})$ and being rewarded according to V^t in the next step. In the formula above we use $\oplus$ and $\otimes$ to indicate that the addition and multiplication operations are performed on functions over relational structures. Each such function defines a partition over the state space as described above and assigns a value or probability to states in the state space.

3. **Object Maximization:** Maximize over the action parameters of $Q_{V^t}^{A(\vec{x})}$ to produce $Q_{V^t}^{A}$ for each action $A(\vec{x})$, thus obtaining the value achievable by the best ground instantiation of $A(\vec{x})$.

4. **Maximize over Actions:** The $t+1$-step-to-go value function is generated by maximizing over all actions, $V^{t+1} = \max_A Q_{V^t}^{A}$.

In this description of RMDPs all intermediate constructs (R, P, V etc.) are represented in some compact form and they capture a mapping from states to values or probabilities. The operations of the Bellman update are performed over these functions while maintaining the compact form.

The advantage of the relational representation is abstraction. One can plan at the abstract level without grounding the domain, potentially leading to more efficient algorithms. In addition, the solution at the abstract level is optimal for every instantiation of the domain and can be reused for multiple problems. However, this approach raises some difficult computational issues because one must use theorem proving to reason at the abstract level, and because for some problems optimal solutions at the abstract level can be infinite in size.

Following Boutilier et al. (2001) several abstract versions of the value iteration (VI) algorithm have been developed using different representation schemes. Großmann, Hölldobler, and Skvortsova (2002) developed a SDP algorithm in the context of the fluent calculus. Later Hölldobler and Skvortsova (2004) improved this first order VI algorithm (FOVIA) by employing normalization procedures to remove redundancies from formulas in the fluent calculus. FCPLANNER, a planning system based on FOVIA participated in the International Planning Competition. Kersting et al. (2004) invented a relational Bellman operator based on SDP. Their system, REBEL however, represented the RMDP in a simpler language restricted to existential quantification. This change greatly improved the logical simplification step and hence the planning efficiency. Further REBEL used decision lists (Rivest, 1987) to represent value functions. Since a decision list implicitly encodes an ordering on the rules in the list, the object maximization step in REBEL was reduced to sorting the decision list. The REBEL paper demonstrated that abstract value functions as generated by SDP based algorithms can be infinite in size. REBEL exhibited excellent performance in solving some case studies. The work of Sanner (2008) and the research presented in this thesis were motivated by the success of ADDs in solving propositional MDPs. While both approaches invent relational versions of ADDs, Sanner (2008) reports on an implementation that does not scale well to large problems. In subsequent chapters we will present a full description of our SDP based RMDP solver along with an implementation that scales to problems from the International Planning Competition.

There are many similarities between the SDP method and previous work on explanation based learning (EBL) (Mitchell, Keller, & Kedar-Cabelli, 1986; DeJong & Mooney, 1986; Laird, Rosenbloom, & Newell, 1986). In the EBL setting the learner has access to positive training examples for a particular concept and a domain theory. The domain theory is sufficient to explain the training examples by itself. The objective of the learner is to generate such explanations for the given training examples and generalize each explanation to produce a hypothesis that can be later used to explain similar examples. Generalization is performed using the same process of regression we discussed in the context of SDP. Since the domain theory already classifies the training examples correctly, the hypotheses generated by EBL does not add any new information but simply compiles existing knowledge into a more readily usable form. Thus the objective of EBL is to speed up the process of classification. This line of research has, therefore, come to be known as speedup learning. EBL developed as an alternative to inductive learning with the advantage that the generated hypotheses are completely justified by the domain theory and hence the error and sample complexity bounds for inductive learning do not apply to EBL.

In deterministic planning domains, the domain theory can be expressed as the description of operators or actions. In such domains the domain theory is guaranteed to be correct and complete and EBL systems can be employed to learn search control rules. Minton (1988) presented such an EBL based system, PRODIGY. PRODIGY was designed to learn a variety of search control rules. These not only included rules about which operators to apply but also rules about ordering parts of the goal. One interesting aspect of PRODIGY was the ability, similar to SDP, to directly reason backwards from the target concept (using regression) without generalizing concrete examples.

Difficult issues that arose in the work on EBL are also seen in SDP based systems. EBL systems faced what is known as the utility problem. In non-trivial domains, the number and complexity of rules learned can be large enough that searching for a solution becomes inefficient. In such cases speedup learning can actually slow down the problem solving process. Minton (1988) addressed this issue by removing rules for which the cost of applying the rule outweighed the benefit of using the rule. Similarly in SDP the value function can becoome prohibitively complex and logical simplification becomes necessary. On the other hand more prunning may be possible for RMDPs because abstract states are associated with values in RMDPs and because all t-step-to-go abstract states are developed simultaneously. Another problem encountered by EBL was known as the generalization to N problem (Shavlik, 1989). Often times hypotheses learned by EBL are specific to some numerical property of the domain (e.g., its size) whereas a solution that generalizes over all values of that numerical property is desired. A similar issue arises in SDP in the form of domains where the value function can be infinite in size. Thus all SDP based solutions to RMDPs encounter these issues.

In the literature there are numerous other representations and solution formalisms for RMDPs. These include approaches that combine dynamic programming with linear function approximation (Sanner & Boutilier, 2009), forward search (Hölldobler, Karabaev, & Skvortsova, 2006) and machine learning (Fern, Yoon, & Givan, 2006; Gretton & Thiebaux, 2004). Sanner and Boutilier (2009), in particular, develop a relational extension of linear function approximation techniques for factored MDPs discussed above. The value function is represented as a weighted sum of basis functions, each denoting a partition of the state space. The difference from the work on factored MDPs is that these basis functions are first order formulas and thus the value function is valid for any domain size (this is the same fundamental advantage that RMDP solvers have over ground MDP solvers). They develop methods for automatic generation of first order constraints in a linear program and automatic generation of basis functions that shows promise in solving some domains from the IPC. The work of Sanner and Boutilier (2009) is thus an extension of the work on linear programming based MDP solvers that takes advantage of relational structure in the domain. In a similar view the work in this thesis is a relational extension of the work on ADD based MDP solvers.

There are also approaches that do not resort to dynamic programming at all. For instance Guestrin, Koller, Gearhart, and Kanodia (2003a) present an approach based on additive value functions based on object classes and employ linear programming to solve the RMDP. Mausam and Weld (2003)'s approach is to employ SPUDD to solve ground instances of an RMDP, generate training data from the solutions and learn a lifted value function from the training data using a relational tree learner. Gardiol and Kaelbling (2003) apply methods from probabilistic planning to solve the RMDP. Yet other RMDP solvers are based on using reinforcement learning techniques. Relational Reinforcement Learning (RRL) (Tadepalli, Givan, & Driessens, 2004) is an active area of research. RRL followed from the seminal work of Dzeroski, De Raedt, and Driessens (2001) whose algorithm involved generating state-action or state-value pairs by state space exploration (biased in favor of state-action pairs with high estimated value) and learning a relational value function tree from the collected data. There have been several approaches to RRL in recent years (Driessens & Dzeroski, 2004; Kersting & De Raedt, 2004; Walker, Torrey, Shavlik, & Maclin, 2007; Croonenborghs, Ramon, Blockeel, & Bruynooghe, 2007). van Otterlo (2008) provides an excellent overview of the various solutions methods to RMDPs.

In the rest of the thesis we present our FODD representation, algorithms for it and its use in solving RMDPs through a relational VI algorithm. To our knowledge, apart from Sanner and Boutilier (2009) and Hölldobler et al. (2006), this is the only dynamic programming based approach to RMDPs that has been shown to scale to problems of the size used in the recent IPCs.

Chapter 3

First Order Decision Diagrams

First Order Decision Diagrams (FODD) are the fundamental unit and underlying thread connecting all the research in this thesis. In this chapter we will describe FODDs and their application in representing and solving relational MDPs. The research presented in this chapter is joint work with Chenggang Wang and has previously appeared in her Ph.D. thesis (Wang, 2008).

This chapter is organised as follows. Sections 3.1 and 3.2 describe the syntax and semantics of FODDs. Sections 3.3 to 3.6 discuss properties of FODDs. Finally Sections 3.7 and 3.8 present an application of FODDs in solving RMDPs.

3.1 Syntax of First Order Decision Diagrams

A decision diagram is a graphical representation for functions over propositional (Boolean) variables. The function is represented as a labeled rooted directed acyclic graph where each non-leaf node is labeled with a propositional variable and has exactly two children. The outgoing edges are marked with values `true` and `false`. Leaves are labeled with numerical values. Given an assignment of truth values to the propositional variables, we can traverse the graph where in each node we follow the outgoing edge corresponding to its truth value. This gives a mapping from any assignment to a leaf of the diagram and in turn to its value. If the leaves are marked with values in $\{0, 1\}$ then we can interpret the graph as representing a Boolean function over the propositional variables. Equivalently, the

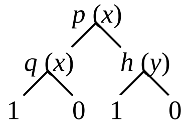


Figure 3.1: A simple FODD.

graph can be seen as representing a logical expression which is satisfied if and only if the 1 leaf is reached. The case with $\{0, 1\}$ leaves is known as Binary Decision Diagrams (BDD) and the case with numerical leaves (or more general algebraic expressions) is known as Algebraic Decision Diagrams (ADD). Decision Diagrams are particularly interesting if we impose an order over propositional variables and require that node labels respect this order on every path in the diagram; this case is known as Ordered Decision Diagrams (ODD). In this case every function has a unique canonical representation that serves as a normal form for the function. This property means that propositional theorem proving is easy for ODD representations. For example, if a formula is contradictory then this fact is evident when we represent it as an ODD, since the normal form for a contradiction is a single leaf valued 0. This property together with efficient manipulation algorithms for ODD representations have led to successful applications, e.g., in VLSI design and verification (Bryant, 1992; McMillan, 1993; Bahar et al., 1993) as well as MDPs (Hoey et al., 1999; St-Aubin et al., 2000). In the following we generalize this representation for relational problems.

There are various ways to generalize ADDs to capture relational structure. One could use closed or open formulas in the nodes, and in the latter case we must interpret the quantification over the variables. In the process of developing the ideas here we have considered several possibilities including explicit quantifiers but these did not lead to useful solutions because of the complexity of manipulating the resultant representations. We therefore focus on the following syntactic definition which does not have any explicit quantifiers. We use standard terminology from first order logic (Lloyd, 1987).

For this representation, we assume a fixed set of predicates and constant symbols, and an enumerable set of variables. We also allow using an equality between any pair of terms (constants or variables).

Definition 1 *First Order Decision Diagram*

1. *A First Order Decision Diagram (FODD) is a labeled rooted directed acyclic graph, where each non-leaf node has exactly two children. The outgoing edges are marked with values* `true` *and* `false`*.*

2. *Each non-leaf node is labeled with: an atom $P(t_1, \ldots, t_n)$ or an equality $t_1 = t_2$ where each t_i is a variable or a constant.*

3. *Leaves are labeled with numerical values.*

Figure 3.1 shows a FODD with binary leaves. Left going edges represent `true` branches. To simplify diagrams in the thesis we draw multiple copies of the leaves 0 and 1 (and occasionally other values or small sub-diagrams) but they represent the same node in the FODD.

We use the following notation: for a node n, $n_{\downarrow t}$ denotes the `true` branch of n, and $n_{\downarrow f}$ the `false` branch of n; $n_{\downarrow a}$ is an outgoing edge from n, where a can be `true` or `false`. For an edge e, $source(e)$ is the node that edge e issues from, and $target(e)$ is the node that edge e points to. Let e_1 and e_2 be two edges, we have $e_1 = sibling(e_2)$ iff $source(e_1) = source(e_2)$.

In the following we will slightly abuse the notation and let $n_{\downarrow a}$ mean either an edge or the sub-FODD this edge points to. We will also use $n_{\downarrow a}$ and $target(e_1)$ interchangeably where $n = source(e_1)$ and a can be `true` or `false` depending on whether e_1 lies in the `true` or `false` branch of n.

3.2 Semantics of First Order Decision Diagrams

We use a FODD to represent a function that assigns values to states in a relational MDP. For example, in the logistics domain, we might want to assign values to different states in such a way that if there is a box in Paris, then the state is assigned a value of 19; if there is no box in Paris but there is a box on a truck that is in Paris and it is raining, this state is assigned a value of 6.3[1]. The question is how to define the semantics of FODDs in order to have the intended meaning.

[1] This is a result of regression in the logistics domain; cf. Figure 3.13(l).

The semantics of first order formulas are given relative to interpretations. An interpretation has a domain of elements, a mapping of constants to domain elements and, for each predicate, a relation over the domain elements which specifies when the predicate is true. There is more than one way to define the meaning of FODD B on interpretation I.

We build on work by Groote and Tveretina (2003) who defined semantics based on multiple paths. Following this work, we define the semantics first relative to a variable valuation ζ. Given a FODD B over variables $\vec{x}$ and an interpretation I, a valuation ζ maps each variable in $\vec{x}$ to a domain element in I. Once this is done, each node predicate evaluates either to `true` or `false` and we can traverse a single path to a leaf. The value of this leaf is denoted by $\text{MAP}_B(I, \zeta)$.

Different valuations may give different values; but recall that we use FODDs to represent a function over states, and each state must be assigned a single value. Therefore, we next define

$$\text{MAP}_B(I) = \text{aggregate}_\zeta\{\text{MAP}_B(I, \zeta)\}$$

for some aggregation function. That is, we consider all possible valuations ζ, and for each valuation we calculate $\text{MAP}_B(I, \zeta)$. We then aggregate over all these values. In the special case of Groote and Tveretina (2003) leaf labels are in $\{0, 1\}$ and variables are universally quantified; this is easily captured in our formulation by using minimum as the aggregation function. For FODDs, we use maximum as the aggregation function.

$$\text{MAP}_B(I) = \max_\zeta\{\text{MAP}_B(I, \zeta)\}.$$

This corresponds to existential quantification in the binary case (if there is a valuation leading to value 1, then the value assigned will be 1) and gives useful maximization for value functions in the general case. Using this definition B assigns every I a unique value $v = \text{MAP}_B(I)$ so B defines a function from interpretations to real values. We later refer to this function as *the map of* B.

Consider evaluating the diagram in Figure 3.1 on the interpretation I_1 where the only true atoms are $\{p(1), q(2), h(3)\}$. The valuation where x is mapped to 2 and y is mapped to 3 denoted $\{x/2, y/3\}$ leads to a leaf with value 1. Because there exists a valuations reaching the 1 leaf, the maximum over the values of leaves reached by all valuations is 1.

When leaf labels are in $\{0, 1\}$, we can interpret the diagram as a logical formula. When $\text{MAP}_B(I) = 1$, as in our example, we say that I satisfies B and when $\text{MAP}_B(I) = 0$ we say that I falsifies B.

3.3 Basic Reduction of FODDs

Groote and Tveretina (2003) define several operators that reduce a diagram into normal form. A total order over node labels is assumed. We describe these operators briefly and give their main properties.

(R1) Neglect operator: if both children of a node p in the FODD lead to the same node q then we remove p and link all parents of p to q directly.

(R2) Join operator: if two nodes p, q have the same label and point to the same two children then we can join p and q (remove q and link q's parents to p).

(R3) Merge operator: if a node and its child have the same label then the parent can point directly to the grandchild of the parent.

(R4) Sort operator: If a node p is a parent of q but the label ordering is violated ($l(q) \prec l(p)$) then we can reorder the nodes locally using two copies of p and q such that labels of the nodes do not violate the ordering.

Define a FODD to be reduced if none of the four operators can be applied. Groote and Tveretina (2003) have shown the following:

Theorem 1 *(Groote & Tveretina, 2003)*
(1) Let $O \in \{$Neglect, Join, Merge, Sort$\}$ be an operator and $O(B)$ the result of applying O to FODD B, then for any B, I, and ζ, MAP$_B(I, \zeta)$ = MAP$_{O(B)}(I, \zeta)$.
(2) If B_1, B_2 are reduced and satisfy $\forall\zeta$, MAP$_{B_1}(I, \zeta)$ = MAP$_{B_2}(I, \zeta)$ then they are identical.

Property (1) gives soundness, and property (2) shows that reducing a FODD gives a normal form. However, this only holds if the maps are identical for every ζ and this condition is stronger than equivalence of the maps. This normal form suffices for Groote and Tveretina (2003) who use it to provide a theorem prover for first order logic, but it is not strong enough for our purposes. Figure 3.2 shows two pairs of reduced FODDs (with respect to R1-R4) such that $\text{MAP}_{B_1}(I) = \text{MAP}_{B_2}(I)$ but $\exists\zeta, \text{MAP}_{B_1}(I,\zeta) \neq \text{MAP}_{B_2}(I,\zeta)$. In this case although the maps are the same the FODDs are not reduced to the same form. Consider first the pair in the upper part of the figure. An interpretation where $p(a)$ is false but $p(b)$ is true and and a substitution $\{x/a, y/b\}$ leads to value of 0 in B_1 while B_2 always evaluates to 1. But the diagrams are equivalent. For any interpretation, if $p(c)$ is true for any object c then $\text{MAP}_{B_1}(I) = 1$ through the substitution $\{x/c\}$; if $p(c)$ is false for any object c then $\text{MAP}_{B_1}(I) = 1$ through the substitution $\{x/c, y/c\}$. Thus the map is always 1 for B_1 as well. In Section 3.6.2 we show that with the additional reduction operators we have developed, B_1 in the first pair is reduced to 1. Thus the diagrams in the upper part have the same form after reduction. However, our reductions do not resolve the pair given in the lower part of the figure. Notice that both functions capture a path of two edges labeled p in a graph (we just change the order of two nodes and rename variables) so the diagrams evaluate to 1 if and only if the interpretation has such a path. Even though B_1 and B_2 are logically equivalent, they cannot be reduced to the same form using R1-R4 or our new operators. To identify a unique minimal syntactic form one may have to consider all possible renamings of variables and the sorted diagrams they produce, but this is an expensive operation. A discussion of normal form for conjunctions that uses such an operation is given by Garriga, Khardon, and De Raedt (2007).

3.4 Combining FODDs

Given two algebraic diagrams we may need to add the corresponding functions, take the maximum or use any other binary operation, `op`, over the values represented by the functions. Here we adopt the solution from the propositional case (Bryant, 1986) in the form of the procedure **Apply(B_1,B_2,`op`)** defined as follows.

Definition 2 *Let B = Apply(B_1,B_2,`op`) and let p and q be the roots of B_1 and B_2 respectively.*

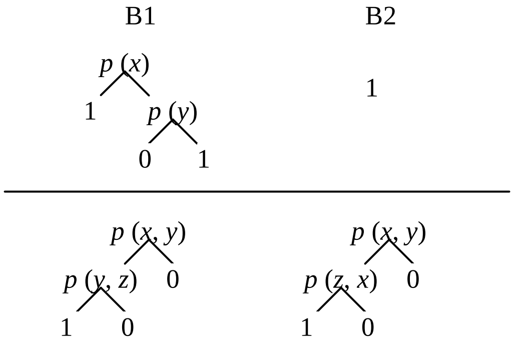


Figure 3.2: Examples illustrating weakness of normal form.

1. *If p and q are both leaves, $B = p$ `op` q.*

2. *If p preceeds q according to the order of the labels, then p is the root of B, the left sub-diagram of B is given by Apply($p_{\downarrow t}$,B_2,`op`) and the right sub-diagram of B is given by Apply($p_{\downarrow f}$,B_2,`op`).*

3. *If q preceeds p according to the order of the labels, then q is the root of B, the left sub-diagram of B is given by Apply(B_1,$q_{\downarrow t}$,`op`) and the right sub-diagram of B is given by Apply(B_1,$q_{\downarrow f}$,`op`).*

4. *If $p = q$, then p (or q) is the root of B, the left sub-diagram of B is given by Apply($p_{\downarrow t}$,$q_{\downarrow t}$,`op`) and the right sub-diagram of B is given by Apply($p_{\downarrow f}$,$q_{\downarrow f}$,`op`).*

This procedure chooses a new root label (the lower among labels of p, q) and recursively combines the corresponding sub-diagrams, according to the relation between the two labels ($\prec$, $=$, or $\succ$). In order to make sure the result is reduced in the propositional sense one can use dynamic programming to avoid generating nodes for which either neglect or join operators ((R1) and (R2) above) would be applicable.

Figure 3.3 illustrates this process. In this example, we assume predicate ordering as $p_1 \prec p_2$, and parameter ordering $x_1 \prec x_2$. Non-leaf nodes are annotated with numbers and numerical leaves are underlined for identification during the execution trace. For example, the top level call adds the functions corresponding to nodes 1 and 3. Since $p_1(x_1)$ is the smaller label it is picked as the label for the root of the result. Then we must add both left

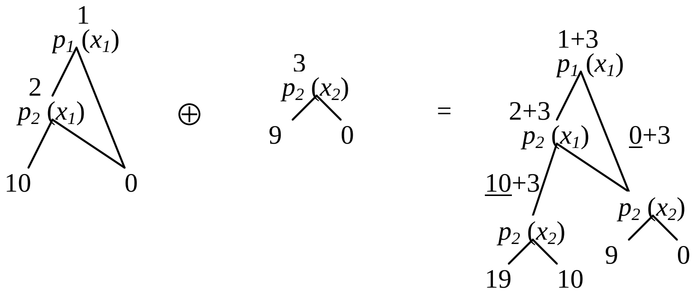


Figure 3.3: A simple example of adding two FODDs.

and right child of node 1 to node 3. These calls are performed recursively. It is easy to see that the size of the result may be the product of sizes of input diagrams. However, pruning will occur with shared variables and further pruning is made possible by weak reductions presented later.

Since for any interpretation I and any fixed valuation ζ the FODD is propositional, we have the following. We later refer to this property as the *correctness of* **Apply**.

Lemma 1 *Let $C = Apply(A, B, op)$, then for any I and ζ, MAP$_A(I, \zeta)$ op MAP$_B(I, \zeta) =$ MAP$_C(I, \zeta)$.*

Proof: First we introduce some terminology. Let $\#nodes(X)$ refer to the set of all nodes in a FODD X. Let the root nodes of A and B be A_{root} and B_{root} respectively. Let the FODDs rooted at $A_{root_{\downarrow t}}$, $A_{root_{\downarrow f}}$, $B_{root_{\downarrow t}}$, $B_{root_{\downarrow f}}$, $C_{root_{\downarrow t}}$, and $C_{root_{\downarrow f}}$ be A^l, A^r, B^l, B^r, C^l and C^r respectively.

The proof is by induction on $n = |\#nodes(A)| + |\#nodes(B)|$. The lemma is true for $n = 2$, because in this case both A_{root} and B_{root} have to be single leaves and an operation on them is the same as an operation on two real numbers. For the inductive step we need to consider two cases.

Case 1: $A_{root} = B_{root}$. Since the root nodes are equal, if a valuation ζ reaches A^l, then it will also reach B^l and if ζ reaches A^r, then it will also reach B^r. Also, by the definition of Apply, in this case $C^l = Apply(A^l, B^l, op)$ and $C^r = Apply(A^r, B^r, op)$. Therefore the statement of the lemma is true if MAP$_{A^l}(I, \zeta)$ op MAP$_{B^l}(I, \zeta) =$ MAP$_{C^l}(I, \zeta)$ and

MAP$_{A^r}(I,\zeta)$ op MAP$_{B^r}(I,\zeta) =$ MAP$_{C^r}(I,\zeta)$ for any ζ and I. Now, since $|\#nodes(A^l)+\#nodes(B^l)| < n$ and $|\#nodes(A^r) + \#nodes(B^r)| < n$, this is guaranteed by the induction hypothesis.

Case 2: $A_{root} \neq B_{root}$. Without loss of generality let us assume that $A_{root} \prec B_{root}$. By the definition of Apply, $C^l = Apply(A^l, B, op)$ and $C^r = Apply(A^r, B, op)$. Therefore the statement of the lemma is true if MAP$_{A^l}(I,\zeta)$ op MAP$_B(I,\zeta) =$ MAP$_{C^l}(I,\zeta)$ and MAP$_{A^r}(I,\zeta)$ op MAP$_B(I,\zeta) =$ MAP$_{C^r}(I,\zeta)$ for any ζ and I. Again this is guaranteed by the induction hypothesis. ■

3.5 Order of Labels

The syntax of FODDs allows for two "types" of objects: constants and variables. Any argument of a predicate can be a constant or a variable. We assume a complete ordering on predicates, constants, and variables. The ordering $\prec$ between two labels is given by the following rules.

1. $P(x_1, ..., x_n) \prec P'(x'_1, ..., x'_m)$ if $P \prec P'$
2. $P(x_1, ..., x_n) \prec P(x'_1, ..., x'_n)$ if there exists i such that $x_j = x'_j$ for all $j < i$, and $type(x_i) \prec type(x'_i)$ (where "type" can be constant or variable) or $type(x_i) = type(x'_i)$ and $x_i \prec x'_i$.

While the predicate order can be set arbitrarily it appears useful to assign the equality predicate as the first in the predicate ordering so that equalities are at the top of the diagrams. During reductions we often encounter situations where one side of the equality can be completely removed leading to substantial space savings. It may also be useful to order the argument types so that constant $\prec$ variables. This ordering may be helpful for reductions. Intuitively, a variable appearing lower in the diagram can be bound to the value of a constant that appears above it. These are only heuristic guidelines and the best ordering may well be problem dependent. We later introduce other forms of arguments: *predicate parameters* and *action parameters*. The ordering for these is discussed in Section 3.8.

3.6 Reduction Operators

In our context, especially for algebraic FODDs, we may want to reduce the diagrams beyond the compression achieved by R1-R4. We distinguish between *strong reductions* that preserve MAP$_B(I, \zeta)$ for all ζ and *weak reductions* that only preserve MAP$_B(I)$. Theorem 1 shows that R1-R4 given above are strong reductions. In the following we present the details of R5 and R7 that are used in our implementation. These details are relevant in Chapter 4 where some improvements are presented. For a discussion of R6 and R8 see Wang (2008).

All the reduction operators below can incorporate existing knowledge on relationships between predicates in the domain. We denote this background knowledge by $\mathcal{B}$. For example in the Blocks World we may know that if there is a block on block y then it is not clear: $\forall x, y, [on(x, y) \rightarrow \neg clear(y)]$.

When we define conditions for reduction operators, there are two types of conditions: the reachability condition and the value condition. We name reachability conditions by starting with P (for Path Condition) and the reduction operator number. We name conditions on values by starting with V and the reduction operator number.

In the following we define node formulas (NF) and edge formulas (EF) recursively as follows. For a node n labeled $l(n)$ with incoming edges $e_1, \ldots, e_k$, the node formula NF$(n) = (\vee_i$EF$(e_i))$. The edge formula for the `true` outgoing edge of n is EF$(n_{\downarrow t}) =$ NF$(n) \wedge l(n)$. The edge formula for the `false` outgoing edge of n is EF$(n_{\downarrow f}) =$ NF$(n) \wedge \neg l(n)$. These formulas, where all variables are existentially quantified, capture the conditions under which a node or edge are reached.

3.6.1 (R5) Strong Reduction for Implied Branches

Consider any node n such that whenever n is reached then the `true` branch is followed. In this case we can remove n and connect its parents directly to the `true` branch. We first present the condition, followed by the lemma regarding this operator.

(P5) : $\mathcal{B} \models \forall \vec{x}, [NF(n) \rightarrow l(n)]$ where $\vec{x}$ are the variables in EF$(n_{\downarrow t})$.

Let R5(n) denote the operator that removes node n and connects its parents directly to the `true` branch. Notice that this is a generalization of R3. It is easy to see that the following lemma is true:

Lemma 2 *(Wang, 2008) Let B be a FODD, n a node for which condition P5 holds, and B' the result of R5(n). Then for any interpretation I and any valuation ζ we have MAP$_B(I,\zeta) =$ MAP$_{B'}(I,\zeta)$.*

A similar reduction can be formulated for the `false` branch, i.e., if $\mathcal{B} \models \forall \vec{x}, [\text{NF}(n) \rightarrow \neg l(n)]$ then whenever node n is reached then the `false` branch is followed. In this case we can remove n and connect its parents directly to the `false` branch.

Implied branches may simply be a result of equalities along a path. For example $(x = y) \wedge p(x) \rightarrow p(y)$ so we may prune $p(y)$ if $(x = y)$ and $p(x)$ are known to be true. Implied branches may also be a result of background knowledge. For example in the Blocks World if $on(x, y)$ is guaranteed to be true when we reach a node labeled $clear(y)$ then we can remove $clear(y)$ and connect its parent to $clear(y)_{\downarrow f}$.

3.6.2 (R7) Weak Reduction Removing Dominated Edges

Consider any two edges e_1 and e_2 in a FODD whose formulas satisfy the condition that if we can follow e_2 using some valuation then we can also follow e_1 using a possibly different valuation. If e_1 gives better value than e_2 then intuitively e_2 never determines the value of the diagram and is therefore redundant. We formalize this as reduction operator R7.

Let $p = source(e_1), q = source(e_2)$, $e_1 = p_{\downarrow a}$, and $e_2 = q_{\downarrow b}$, where a and b can be `true` or `false`. We first present all the conditions for the operator and then follow with the definition of the operator.

(P7.1) : $\mathcal{B} \models [\exists \vec{x}, \text{EF}(e_2)] \rightarrow [\exists \vec{y}, \text{EF}(e_1)]$ where $\vec{x}$ are the variables in EF(e_2) and $\vec{y}$ the variables in EF(e_1).

(P7.2) : $\mathcal{B} \models \forall \vec{u}, [[\exists \vec{w}, \text{EF}(e_2)] \rightarrow [\exists \vec{v}, \text{EF}(e_1)]]$ where $\vec{u}$ are the variables that appear in both $target(e_1)$ and $target(e_2)$, $\vec{v}$ the variables that appear in EF(e_1) but are not in $\vec{u}$, and $\vec{w}$ the variables that appear in EF(e_2) but are not in $\vec{u}$. This condition requires that for every valuation ζ_1 that reaches e_2 there is a valuation ζ_2 that reaches e_1 such that ζ_1 and ζ_2 agree on all variables that appear in both $target(e_1)$ and $target(e_2)$.

(P7.3) : $\mathcal{B} \models \forall \vec{r}, [[\exists \vec{s}, \text{EF}(e_2)] \rightarrow [\exists \vec{t}, \text{EF}(e_1)]]$ where $\vec{r}$ are the variables that appear in both $target(e_1)$ and $target(sibling(e_2))$, $\vec{t}$ the variables that appear in EF(e_1) but are not in $\vec{r}$, and $\vec{s}$ the variables that appear in EF(e_2) but are not in $\vec{r}$. This condition requires that for every valuation ζ_1 that reaches e_2 there is a valuation ζ_2 that reaches e_1 such that ζ_1 and ζ_2 agree on all variables that appear in both $target(e_1)$ and $target(sibling(e_2))$.

(V7.1) : $\min(target(e_1)) \geq \max(target(e_2))$ where $\min(target(e_1))$ is the minimum leaf value in $target(e_1)$, and $\max(target(e_2))$ the maximum leaf value in $target(e_2)$. In this case regardless of the valuation we know that it is better to follow e_1 and not e_2.

(V7.2) : $\min(target(e_1)) \geq \max(target(sibling(e_2)))$.

(V7.3) : all leaves in $D = target(e_1) \ominus target(e_2)$ have non-negative values, denoted as $D \geq 0$. In this case for any fixed valuation it is better to follow e_1 instead of e_2.

(V7.4) : all leaves in $G = target(e_1) \ominus target(sibling(e_2))$ have non-negative values.

We define the operators R7-replace(b, e_1, e_2) as replacing $target(e_2)$ with a constant b that is between 0 and $\min(target(e_1))$ (we may write it as R7-replace(e_1, e_2) if $b = 0$), and R7-drop(e_1, e_2) as dropping the node $q = source(e_2)$ and connecting its parents to $target(sibling(e_2))$.

We need one more "safety" condition to guarantee that the reduction is correct:

(S1) : NF$(source(e_1))$ and the sub-FODD of $target(e_1)$ remain the same before and after R7-replace and R7-drop. This condition says that we must not harm the value promised by $target(e_1)$. In other words, we must guarantee that $p = source(e_1)$ is reachable just as before and the sub-FODD of $target(e_1)$ is not modified after replacing a branch with 0. The condition is violated if q is in the sub-FODD of $p_{\downarrow a}$, or if p is in the sub-FODD of $q_{\downarrow b}$. But it holds in all other cases, that is when p and q are unrelated (one is not the descendant of the other), or q is in the sub-FODD of $p_{\downarrow \overline{a}}$, or p is in the sub-FODD of $q_{\downarrow \overline{b}}$, where $\overline{a}, \overline{b}$ stand for negation. The following results guarantee correctness of the reduction.

Lemma 3 *(Wang, 2008) Let B be a FODD, e_1 and e_2 edges for which conditions P7.1, V7.1, and S1 hold, and B' the result of R7-replace(b, e_1, e_2), where $0 \leq b \leq \min(target(e_1))$, then for any interpretation I we have MAP$_B(I)$ = MAP$_{B'}(I)$.*

Lemma 4 *(Wang, 2008) Let B be a FODD, e_1 and e_2 edges for which conditions P7.2, V7.3, and S1 hold, and B' the result of R7-replace(b, e_1, e_2), where $0 \leq b \leq \min(target(e_1))$, then for any interpretation I we have MAP$_B(I)$ = MAP$_{B'}(I)$.*

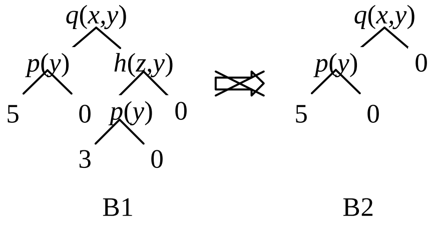


Figure 3.4: An example illustrating the subtraction condition in R7.

Note that the conditions in the previous two lemmas are not comparable since P7.2 → P7.1 and V7.1 → V7.3. Intuitively when we relax the conditions on values, we need to strengthen the conditions on reachability. The subtraction operation $D = target(e_1) \ominus target(e_2)$ is propositional, and hence the test in V7.3 implicitly assumes that the common variables in the operands are the same and P7.1 does not check this. Figure 3.4 illustrates that the reachability condition P7.1 together with V7.3, i.e., combining the weaker portions of conditions from Lemma 3 and Lemma 4, cannot guarantee that we can replace a branch with a constant. Consider an interpretation I with domain $\{1, 2, 3, 4\}$ and relations $\{h(1, 2), q(3, 4), p(2)\}$. In addition assume domain knowledge $\mathcal{B} = [\exists x, y, h(x, y) \rightarrow \exists z, w, q(z, w)]$. Therefore P7.1 and V7.3 hold for $e_1 = [q(x, y)]_{\downarrow t}$ and $e_2 = [h(z, y)_{\downarrow t}]$. We have MAP$_{B1}(I) = 3$ and MAP$_{B2}(I) = 0$. It is therefore not possible to replace $h(z, y)_{\downarrow t}$ with 0.

Sometimes we can drop the node q completely with R7-drop. Intuitively, when we remove a node, we must guarantee that we do not gain extra value. The conditions for R7-replace can only guarantee that we will not lose any value. But if we remove the node q, a valuation that was supposed to reach e_2 may reach a better value in e_2's sibling. This would change the map, as illustrated in Figure 3.5. Notice that the conditions P7.1 and V7.1 hold for $e_1 = [p(x)]_{\downarrow t}$ and $e_2 = [p(y)]_{\downarrow t}$ so we can replace $[p(y)]_{\downarrow t}$ with a constant. Consider an interpretation I with domain $\{1, 2\}$ and relations $\{q(1), p(2), h(2)\}$. We have MAP$_{B1}(I) = 10$ via valuation $\{x/2\}$ and MAP$_{B2}(I) = 20$ via valuation $\{x/1, y/2\}$. Thus removing $p(y)$ is not correct.

Therefore we need the additional condition to guarantee that we will not gain extra

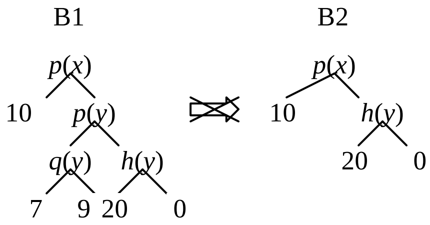


Figure 3.5: An example illustrating the condition for removing a node in R7.

value with node dropping. This condition can be stated as: for any valuation ζ_1 that reaches e_2 and thus will be redirected to reach a value v_1 in $sibling(e_2)$ when q is removed, there is a valuation ζ_2 that reaches a leaf with value $v_2 \geq v_1$. However, this condition is too complex to test in practice. In the following we identify two stronger conditions.

Lemma 5 *(Wang, 2008) Let B be a FODD, e_1 and e_2 edges for which condition V7.2 hold in addition to the conditions for replacing $target(e_2)$ with a constant, and B' the result of R7-drop(e_1, e_2), then for any interpretation I we have MAP$_B(I) =$ MAP$_{B'}(I)$.*

Lemma 6 *(Wang, 2008) Let B be a FODD, e_1 and e_2 edges for which P7.3 and V7.4 hold in addition to conditions for replacing $target(e_2)$ with a constant, and B' the result of R7-drop(e_1, e_2), then for any interpretation I we have MAP$_B(I) =$ MAP$_{B'}(I)$.*

To summarize if P7.1 and V7.1 and S1 hold or P7.2 and V7.3 and S1 hold then we can replace $target(e_2)$ with a constant. If we can replace and V7.2, or both P7.3 and V7.4 hold, then we can drop $q = source(e_2)$ completely. These conditions are simplified in the next chapter.

In some cases, several instances of R7 are applicable and one has to choose which instance to apply first. This is an interesting issue because it turns out that the order in which we apply them is important. In some cases the order can affect the number of steps needed to reduce the diagram. In other cases it can affect the final result. A discussion and examples of these issues is given by Wang (2008).

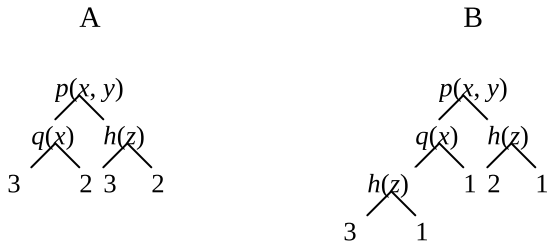


Figure 3.6: An example illustrating that the minimal set of variables for subtraction is not unique.

Relaxation of Reachability Conditions

The conditions P7.2 and P7.3 are sufficient, but not necessary to guarantee correct reductions. Sometimes valuations just need to agree on a smaller set of variables than the intersection of variables. To see this, consider the example as shown in Figure 3.6, where $A \ominus B > 0$ and the intersection is $\{x, y, z\}$. However, to guarantee $A \ominus B > 0$ we just need to agree on either $\{x, y\}$ or $\{x, z\}$. Intuitively we have to agree on the variable x to avoid the situation when two paths $p(x, y) \wedge \neg q(x)$ and $p(x, y) \wedge q(x) \wedge h(z)$ can co-exist. In order to prevent the co-existence of two paths $\neg p(x, y) \wedge \neg h(z)$ and $p(x, y) \wedge q(x) \wedge h(z)$, either y or z has to be the same as well. Now if we change this example a little and replace each $h(z)$ with $h(z, v)$, then we have two minimal sets of variables of different size, one is $\{x, y\}$, and the other is $\{x, z, v\}$. As a result we cannot identify a minimum set of variables for the subtraction and must either choose the intersection or heuristically identify a minimal set, for example, using a greedy procedure. In chapter 4 we will present one such procedure that works well in practice.

3.7 Decision Diagrams for MDPs

In this section we show how FODDs can be used to capture an RMDP. We therefore use FODDs to represent the domain dynamics of deterministic action alternatives, the probabilistic choice of action alternatives, the reward function, and value functions.

3.7.1 Example Domain

We first give a concrete formulation of the logistics problem discussed in the introduction. This example follows exactly the details given by Boutilier et al. (2001), and is used to illustrate our constructions for MDPs. The domain includes boxes, trucks and cities, and predicates are $Bin(Box, City)$, $Tin(Truck, City)$, and $On(Box, Truck)$. Following Boutilier et al. (2001), we assume that $On(b, t)$ and $Bin(b, c)$ are mutually exclusive, so a box on a truck is not in a city and vice versa. That is, our background knowledge includes statements $\forall b, c, t, On(b, t) \rightarrow \neg Bin(b, c)$ and $\forall b, c, t, Bin(b, c) \rightarrow \neg On(b, t)$. The reward function, capturing a planning goal, awards a reward of 10 if the formula $\exists b, Bin(b, Paris)$ is true, that is if there is any box in Paris. Thus the reward is allowed to include constants but need not be completely ground.

The domain includes 3 actions $load, unload$, and $drive$. Actions have no effect if their preconditions are not met. Actions can also fail with some probability. When attempting $load$, a successful version $loadS$ is executed with probability 0.99, and an unsuccessful version $loadF$ (effectively a no-operation) with probability 0.01. The drive action is executed deterministically. When attempting $unload$, the probabilities depend on whether it is raining or not. If it is not raining then a successful version $unloadS$ is executed with probability 0.9, and $unloadF$ with probability 0.1. If it is raining $unloadS$ is executed with probability 0.7, and $unloadF$ with probability 0.3.

3.7.2 The Domain Dynamics

We follow Boutilier et al. (2001) and specify stochastic actions as a randomized choice among deterministic alternatives. The domain dynamics are defined by *truth value diagrams* (TVDs). For every action schema $A(\vec{a})$ and each predicate schema $p(\vec{x})$ the TVD $T(A(\vec{a}), p(\vec{x}))$ is a FODD with $\{0, 1\}$ leaves. The TVD gives the truth value of $p(\vec{x})$ in the next state when $A(\vec{a})$ has been performed in the current state. We call $\vec{a}$ action parameters, and $\vec{x}$ predicate parameters. No other variables are allowed in the TVD; the reasoning behind this restriction is explained in Section 3.8.1. The restriction can be sometimes sidestepped by introducing more action parameters instead of the variables.

The TVD simultaneously captures the truth values of all instances of $p(\vec{x})$ in the next

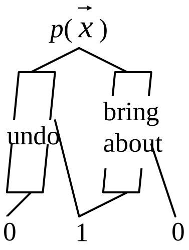


Figure 3.7: A template for the TVD

state. Notice that TVDs for different predicates are separate. This can be safely done even if an action has coordinated effects (not conditionally independent) since the action alternatives are deterministic.

Because we allow both action parameters and predicate parameters, the effects of an action are not restricted to predicates over action arguments so TVD are more expressive than simple STRIPS based schemas. For example, TVDs can easily express universal effects of an action. To see this note that if $p(\vec{x})$ is true for all x after action $A(\vec{a})$ then the TVD $T(A(\vec{a}), p(\vec{x}))$ can be captured by a leaf valued 1. Other universal conditional effects can be captured similarly. On the other hand, because we do not have explicit universal quantifiers, TVDs cannot capture universal preconditions.

For any domain, a TVD for predicate $p(\vec{x})$ can be defined generically as in Figure 3.7. The idea is that the predicate is true if it was true before and is not “undone” by the action or was false before and is “brought about” by the action. TVDs for the logistics domain in our running example are given in Figure 3.8. All the TVDs omitted in the figure are trivial in the sense that the predicate is not affected by the action. In order to simplify the presentation we give the TVDs in their generic form and did not sort the diagrams using the order proposed in Section 3.5; the TVDs are consistent with the ordering $Bin \prec$ “=” $\prec On \prec Tin \prec rain$. Notice that the TVDs capture the implicit assumption usually taken in such planning-based domains that if the preconditions of the action are not satisfied then the action has no effect.

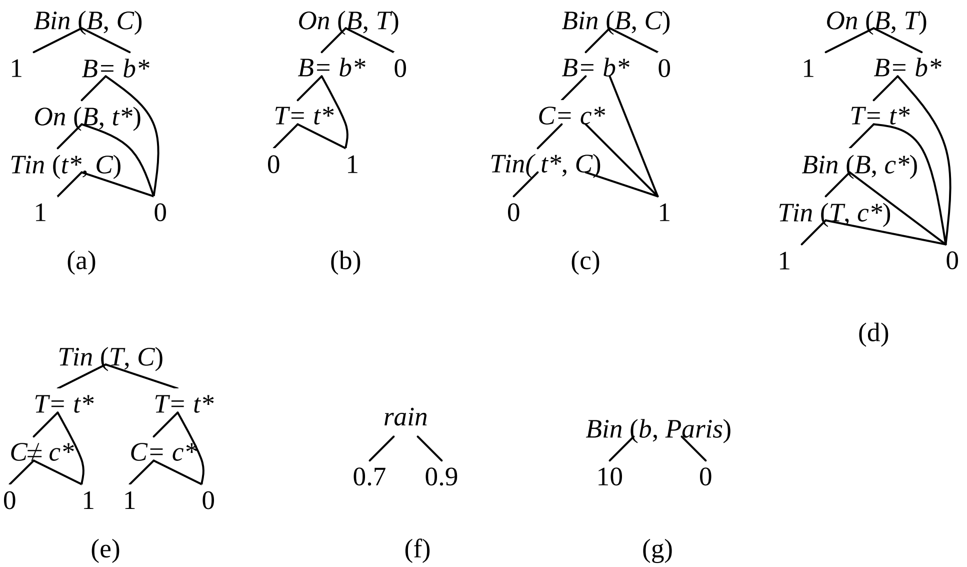


Figure 3.8: FODDs for the logistics domain: TVDs, action choice, and reward function. (a)(b) The TVDs for $Bin(B,C)$ and $On(B,T)$ under action choice $unloadS(b^*,t^*,c^*)$. (c)(d) The TVDs for $Bin(B,C)$ and $On(B,T)$ under action choice $loadS(b^*,t^*,c^*)$. Note that c^* must be an action parameter so that (d) is a valid TVD. (e) The TVD for $Tin(T,C)$ under action choice $driveS(t^*,c^*)$. (f) The probability FODD for the action choice $unloadS(b^*,t^*)$. Therefore, $P(unloadS(b^*,t^*,c^*) \mid unload(b^*,t^*,c^*), rain) = 0.7$ and $P(unloadS(b^*,t^*,c^*) \mid unload(b^*,t^*,c^*), \neg rain) = 0.9$. (g) The reward function.

3.7.3 Probabilistic Action Choice

The probability distribution over action effects can itself depend on the state. One can consider modeling arbitrary conditions described by formulas over the state to control nature's probabilistic choice of action. Here the multiple path semantics makes it hard to specify mutually exclusive conditions using existentially quantified variables and in this way specify a distribution. We therefore restrict the conditions to be either propositional or depend directly on the action parameters. Under this condition any interpretation follows exactly one path (since there are no variables and thus only the empty valuation) thus the aggregation function does not interact with the probabilities assigned. A diagram showing action choice for $unloadS$ in our logistics example is given in Figure 3.8. In this example, the

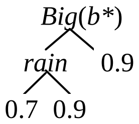


Figure 3.9: An example showing that the choice probability can depend on action parameters.

condition is propositional. The condition can also depend on action parameters, for example, if we assume that the result is also affected by whether the box is big or not, we can have a diagram as in Figure 3.9 specifying the action choice probability.

Note that a probability usually depends on the current state. It can depend on arbitrary properties of the state (with the restriction stated as above), e.g., $rain$ and $big(b^*)$, as shown in Figure 3.9. We allow arbitrary conditions that depend on predicates with arguments restricted to action parameters so the dependence can be complex. However, we do not allow any free variables in the probability choice diagram. For example, we cannot model a probabilistic choice of $unloadS(b^*, t^*, c^*)$ that depends on other boxes on the truck t^*, e.g., $\exists b, On(b, t^*) \wedge b \neq b^* : 0.2$; otherwise, 0.7. While we can write a FODD to capture this condition, the semantics of FODD means that a path to 0.7 will be selected by max aggregation so the distribution cannot be modeled in this way. While this is clearly a restriction, the conditions based on action arguments still give a substantial modeling power.

3.7.4 Reward and Value Functions

Reward and value functions can be represented directly using algebraic FODDs. The reward function for our logistics domain example is given in Figure 3.8.

3.8 Value Iteration with FODDs

Following Boutilier et al. (2001) we define the first order value iteration algorithm as follows: given the reward function R and the action model as input, we set $V_0 = R, n = 0$ and repeat the procedure *Rel-greedy* until termination:

Procedure 1 *Rel-greedy*

1. *For each action type $A(\vec{x})$, compute:*

$$Q^{A(\vec{x})}_{V_n} = R \oplus [\gamma \otimes \oplus_j (prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x})))] \qquad (3.1)$$

2. *$Q^A_{V_n} =$ obj-max$(Q^{A(\vec{x})}_{V_n})$.*
3. *$V_{n+1} = \max_A Q^A_{V_n}$.*

The notation and steps of this procedure were discussed in Chapter 2 except that now $\otimes$ and $\oplus$ work on FODDs instead of case statements. Note that because the reward function does not depend on actions, we can move the object maximization step forward before adding the reward function. I.e., we first have

$$T^{A(\vec{x})}_{V_n} = \oplus_j (prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x}))),$$

followed by

$$Q^A_{V_n} = R \oplus \gamma \otimes \text{obj-max}(T^{A(\vec{x})}_{V_n}).$$

Later we will see that the object maximization step makes more reductions possible; therefore by moving this step forward we get some savings in computation. We compute the updated value function in this way in the comprehensive example of value iteration given later in Section 3.8.7.

Value iteration terminates when $\|V_{i+1} - V_i\| \leq \frac{\varepsilon(1-\gamma)}{2\gamma}$ (Puterman, 1994). In our case we need to test that the values achieved by the two diagrams is within $\frac{\varepsilon(1-\gamma)}{2\gamma}$.

Some formulations of goal based planning problems use an absorbing state with zero additional reward once the goal is reached. We can handle this formulation when there is only one non-zero leaf in R. In this case, we can replace Equation 3.1 with

$$Q^{A(\vec{x})}_{V_n} = max(R, \gamma \otimes \oplus_j (prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x}))).$$

To see why this is correct, note that due to discounting the max value is always $\leq R$. If R is satisfied in a state we do not care about the action (max would be R) and if R is 0 in a state we get the value of the discounted future reward.

Note that we can only do this in goal based domains, i.e., there is only one non-zero leaf. This does not mean that we cannot have disjunctive goals, but it means that we must value each goal condition equally.

3.8.1 Regressing Deterministic Action Alternatives

We first describe the calculation of $Regr(V_n, A_j(\vec{x}))$ using a simple idea we call block replacement. We then proceed to discuss how to obtain the result efficiently.

Consider V_n and the nodes in its FODD. For each such node take a copy of the corresponding TVD, where predicate parameters are renamed so that they correspond to the node's arguments and action parameters are unmodified. BR-regress$(V_n, A(\vec{x}))$ is the FODD resulting from replacing each node in V_n with the corresponding TVD, with outgoing edges connected to the 0, 1 leaves of the TVD.

Recall that an RMDP represents a family of concrete MDPs each generated by choosing a concrete instantiation of the state space (typically represented by the number of objects and their types). The formal properties of our algorithms hold for any concrete instantiation.

Fix any concrete instantiation of the state space. Let s denote a state resulting from executing an action $A(\vec{x})$ in state $\hat{s}$. Notice that V_n and BR-regress$(V_n, A(\vec{x}))$ have exactly the same variables. We have the following lemma:

Lemma 7 *(Wang, 2008) Let ζ be any valuation to the variables of V_n (and thus also the variables of BR-regress$(V_n, A(\vec{x}))$). Then MAP$_{V_n}(s, \zeta) =$ MAP$_{BR-regress(V_n,A(\vec{x}))}(\hat{s}, \zeta)$.*

A naive implementation of block replacement may not be efficient. If we use block replacement for regression then the resulting FODD is not necessarily reduced and moreover, since the different blocks are sorted to start with the result is not even sorted. Reducing and sorting the results may be an expensive operation. Instead we calculate the result as follows. For any FODD V_n we traverse BR-regress$(V_n, A(\vec{x}))$ using postorder traversal in terms of blocks and combine the blocks. At any step we have to combine 3 FODDs such that the parent block has not yet been processed (so it is a TVD with binary leaves) and the two children have been processed (so they are general FODDs). If we call the parent B_n, the `true` branch child B_t and the `false` branch child B_f then we can represent their combination as $[B_n \otimes B_t] \oplus [(1 \ominus B_n) \otimes B_f]$.

Lemma 8 *(Wang, 2008) Let B be a FODD where B_t and B_f are FODDs, and B_n is a FODD with $\{0,1\}$ leaves. Let $\hat{B}$ be the result of using Apply to calculate the diagram $[B_n \otimes B_t] \oplus [(1 \ominus B_n) \otimes B_f]$. Then for any interpretation I and valuation ζ we have MAP$_B(I,\zeta) =$ MAP$_{\hat{B}}(I,\zeta)$.*

A high-level description of the algorithm to calculate BR-regress$(V_n, A(\vec{x}))$ by block combination is as follows:

Procedure 2 *Block Combination for BR-regress$(V_n, A(\vec{x}))$*

1. *Perform a topological sort on V_n nodes (Cormen, Leiserson, Rivest, & Stein, 2001).*

2. *In reverse order, for each non-leaf node n (its children B_t and B_f have already been processed), let B_n be a copy of the corresponding TVD, calculate $[B_n \otimes B_t] \oplus [(1 \ominus B_n) \otimes B_f]$*

3. *Return the FODD corresponding to the root.*

Notice that different blocks share variables so we cannot perform weak reductions during this process. However, we can perform strong reductions in intermediate steps since they do not change the map for any valuation. After the process is completed we can perform any combination of weak and strong reductions since this does not change the map of the regressed value function.

We can now explain why we cannot have variables in TVDs through an example illustrated in Figure 3.10. Suppose we have a value function as defined in Figure 3.10(a), saying that if there is a blue block and a big truck such that the block is not on the truck then value 1 is assigned. Figure 3.10(b) gives the TVD for $On(B,T)$ under action $loadS$, in which c is a variable instead of an action parameter. Figure 3.10(c) gives the result after block replacement. Consider an interpretation $\hat{s}$ with domain $\{b_1, t_1, c_1, c_2\}$ and relations $\{Blue(b_1), Big(t_1), Bin(b_1,c_1), Tin(t_1,c_1)\}$. After the action $loadS(b_1,t_1)$ we will reach the state $s = \{Blue(b_1), Big(t_1), On(b_1,t_1), Tin(t_1,c_1)\}$, which gives us a value of

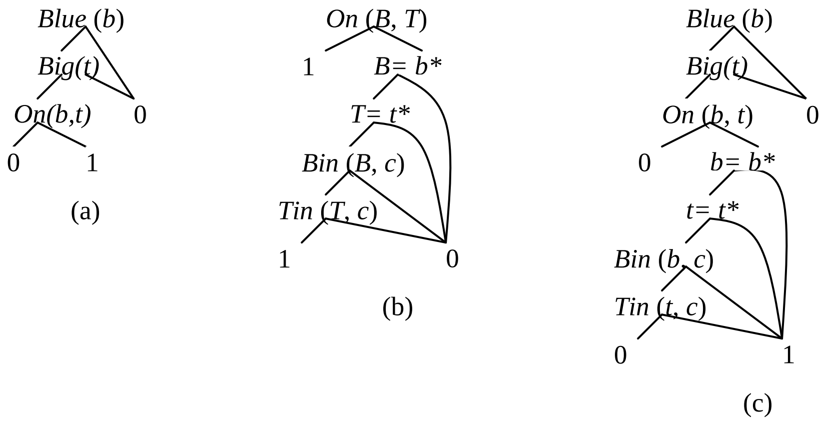


Figure 3.10: An example illustrating why variables are not allowed in TVDs.

0. But Figure 3.10(c) with $b^* = b_1, t^* = t_1$ evaluated in $\hat{s}$ gives value of 1 by valuation $\{b/b_1, c/c_2, t/t_1\}$. Here the choice c/c_2 makes sure the precondition is violated. By making c an action parameter, applying the action must explicitly choose a valuation and this leads to a correct value function. Object maximization turns action parameters into variables and allows us to choose the argument so as to maximize the value.

3.8.2 Regressing Probabilistic Actions

To regress a probabilistic action we must regress all its deterministic alternatives and combine each with its choice probability as in Equation 3.1. As discussed in the previous chapter, due to the restriction in the RMDP model that explicitly specifies a finite number of deterministic action alternatives, we can replace the potentially infinite sum of Equation 2.1 with the finite sum of Equation 3.1 which is therefore correct. In the following we specify how this can be done with FODDs.

Recall that $prob(A_j(\vec{x}))$ is restricted to include only action parameters and cannot include variables. We can therefore calculate $prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x}))$ in step (1) directly using Apply. However, the different regression results are independent functions so that in the sum $\oplus_j(prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x})))$ we must standardize apart the different regression results before adding the functions (note that action parameters are still

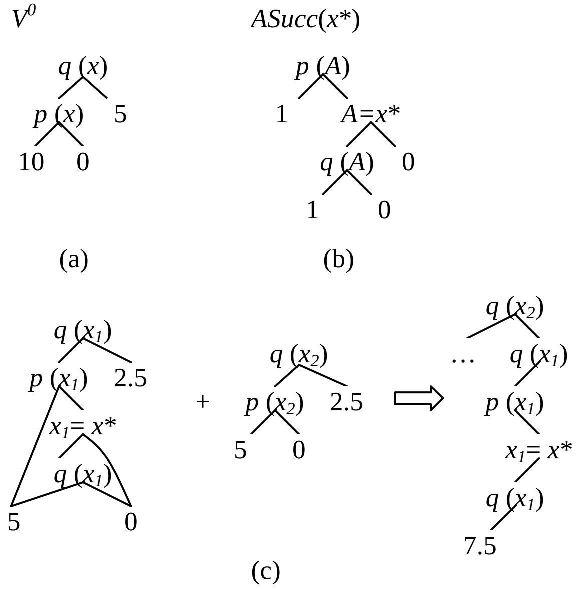


Figure 3.11: An example illustrating the need to standardize apart.

considered constants at this stage). The same holds for the addition of the reward function. The need to standardize apart complicates the diagrams and often introduces structure that can be reduced. When performing these operations we first use the propositional Apply procedure and then follow with weak and strong reductions.

Figure 3.11 illustrates why we need to standardize apart different action outcomes. Action A can succeed (denoted as $ASucc$) or fail (denoted as $AFail$, effectively a no-operation), and each is chosen with probability 0.5. Part (a) gives the value function V^0. Part (b) gives the TVD for $P(A)$ under the action choice $ASucc(x^*)$. All other TVDs are trivial. Part (c) shows part of the result of adding the two outcomes for A after standardizing apart (to simplify the presentation the diagrams are not sorted). Consider an interpretation with domain $\{1, 2\}$ and relations $\{q(1), p(2)\}$. As can be seen from (c), by choosing $x^* = 1$, i.e. action $A(1)$, the valuation $x_1 = 1, x_2 = 2$ gives a value of 7.5 after the action (without considering the discount factor). Obviously if we do not standardize apart (i.e $x_1 = x_2$), there is no leaf with value 7.5 and we get a wrong value. Intuitively the contribution of $ASucc$ to the value comes from the "bring about" portion of the diagram and $AFail$'s contribution uses bindings from the “not undo” portion and the two portions can refer to different objects. Standardizing apart allows us to capture both simultaneously.

From Lemma 7 and 8 and the discussion so far we have:

Lemma 9 *(Wang, 2008) Consider any concrete instantiation of an RMDP. Let V_n be a value function for the corresponding MDP, and let $A(\vec{x})$ be a probabilistic action in the domain. Then $Q_{V_n}^{A(\vec{x})}$ as calculated by Equation 3.1 is correct. That is, for any state s, MAP$_{Q_{V_n}^{A(\vec{x})}}(s)$ is the expected value of executing $A(\vec{x})$ in s and then receiving the terminal value V_n.*

3.8.3 Object Maximization

Notice that since we are handling different probabilistic alternatives of the same action separately we must keep action parameters fixed during the regression process and until they are added in step 1 of the algorithm. In step 2 we maximize over the choice of action parameters. As mentioned above we get this maximization for free. We simply rename the action parameters using new variable names (to avoid repetition between iterations) and consider them as variables. The aggregation semantics provides the maximization and by definition this selects the best instance of the action. Since constants are turned into variables additional reduction is typically possible at this stage. Any combination of weak and strong reductions can be used. From the discussion we have the following lemma:

Lemma 10 *(Wang, 2008) Consider any concrete instantiation of an RMDP. Let V_n be a value function for the corresponding MDP, and let $A(\vec{x})$ be a probabilistic action in the domain. Then $Q_{V_n}^{A}$ as calculated by object maximization in step 2 of the algorithm is correct. That is, for any state s, MAP$_{Q_{V_n}^{A}}(s)$ is the maximum over expected values achievable by executing an instance of $A(\vec{x})$ in s and then receiving the terminal value V_n.*

A potential criticism of our getting object maximization for free is that we are essentially adding more variables to the diagram and thus evaluating the diagram in any state becomes more expensive (since more substitutions need to be considered). However, this is only true if the diagram remains unchanged after object maximization. In fact, as illustrated in the example given below, these variables may be pruned from the diagram in the process of reduction. Thus as long as the final value function is compact the evaluation is efficient and there is no such hidden cost.

3.8.4 Maximizing Over Actions

The maximization $V_{n+1} = \max_A Q^A_{n+1}$ in step (3) combines independent functions. As above we first standardize apart the different diagrams, then we can follow with the propositional Apply procedure and finally follow with weak and strong reductions. This clearly maintains correctness for any concrete instantiation of the state space. The next chapter shows that standardizing apart in this case (and whenever combining diagrams using max) is not necessary.

3.8.5 Order Over Argument Types

We can now resume the discussion of ordering of argument types and extend it to predicate and action parameters. As above some structure is suggested by the operations of the algorithm. Section 3.5 already suggested that we order constants before variables.

Action parameters are “special constants” before object maximization but they become variables during object maximization. Thus their position should allow them to behave as variables. We should therefore order constants before action parameters.

Note that predicate parameters only exist inside TVDs, and will be replaced with domain constants or variables during regression. Thus we only need to decide on the relative order between predicate parameters and action parameters. If we put action parameters before predicate parameters and the latter is replaced with a constant then we get an order violation, so this order is not useful. On the other hand, if we put predicate parameters before action parameters then both instantiations of predicate parameters are possible, as long as the variables corresponding to action parameters (in object maximization) are after the ones in the FODD. Therefore, we also order action parameters after variables.

To summarize, the ordering: constants $\prec$ variables (predicate parameters in case of TVDs) $\prec$ action parameters, is suggested by heuristic considerations for orders that maximize the potential for reductions, and avoid the need for re-sorting diagrams.

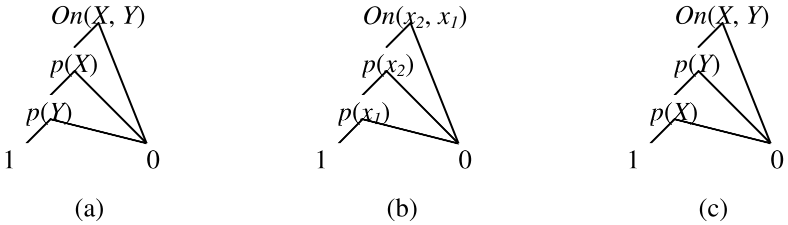


Figure 3.12: An example illustrating the necessity to maintain multiple TVDs.

Finally, note that if we want to maintain the diagram sorted at all times, we need to maintain variant versions of each TVD capturing possible ordering of replacements of predicate parameters. Consider a TVD in Figure 3.12(a). If we rename predicate parameters X and Y to be x_2 and x_1 respectively, and if $x_1 \prec x_2$, then the resulting sub-FODD as shown in Figure 3.12(b) violates the order. To solve this problem we have to define another TVD corresponding to the case where the substitution of X succeeds the substitution of Y, as shown in Figure 3.12(c). In the case of replacing X with x_2 and Y with x_1, we use the TVD in Figure 3.12(c) instead of the one in Figure 3.12(a).

3.8.6 Convergence and Complexity

Since each step of Procedure 1 is correct we have the following theorem:

Theorem 2 *(Wang, 2008) Consider any concrete instantiation of an RMDP. Let V_n be the value function for the corresponding MDP when there are n steps to go. Then the value of V_{n+1} calculated by Procedure 1, correctly captures the value function when there are $n+1$ steps to go. That is, for any state s, MAP$_{V_{n+1}}(s)$ is the maximum expected value achievable in s in $n+1$ steps.*

Note that for first order MDPs some problems require an infinite number of state partitions. Thus we cannot converge to V^* in a finite number of steps. However, since our algorithm implements VI exactly, standard results about approximating optimal value functions and policies still hold. In particular the following standard result (Puterman, 1994) holds for our algorithm, and our stopping criterion guarantees approximating optimal value functions and policies.

Theorem 3 *(Wang, 2008) Let V^* be the optimal value function and let V_k be the value function calculated by the relational VI algorithm.*

1. *If $r(s) \leq M$ for all s then $\|V_n - V^*\| \leq \varepsilon$ for $n \geq \frac{log(\frac{2M}{\varepsilon(1-\gamma)})}{log\frac{1}{\gamma}}$.*
2. *If $\|V_{n+1} - V_n\| \leq \frac{\varepsilon(1-\gamma)}{2\gamma}$ then $\|V_{n+1} - V^*\| \leq \varepsilon$.*

While the algorithm maintains compact diagrams, reduction of diagrams is not guaranteed for all domains. Therefore we can only provide trivial upper bounds in terms of worst case time complexity. Notice first that every time we use the Apply procedure the size of the output diagram may be as large as the product of the size of its inputs. We must also consider the size of the FODD giving the regressed value function. While Block replacement is $O(N)$ where N is the size of the current value function, it is not sorted and sorting may require both exponential time and space in the worst case. For example, Bryant (1986) illustrates how ordering may affect the size of a diagram. For $2n$ arguments, the function $x_1 \cdot x_2 + x_3 \cdot x_4 + \cdots + x_{2n-1} \cdot x_{2n}$ only requires a diagram of $2n + 2$ nodes, while the function $x_1 \cdot x_{n+1} + x_2 \cdot x_{n+2} + \cdots + x_n \cdot x_{2n}$ requires 2^{n+1} nodes. Notice that these two functions only differ by a permutation of their arguments. Now if $x_1 \cdot x_2 + x_3 \cdot x_4 + \cdots + x_{2n-1} \cdot x_{2n}$ is the result of block replacement then clearly sorting requires exponential time and space. The same is true for our block combination procedure or any other method of calculating the result, simply because the output is of exponential size. In such a case heuristics that change variable ordering, as in propositional ADDs (Bryant, 1992), would probably be very useful.

Assuming TVDs, reward function, and probabilities all have size $\leq C$, each action has $\leq M$ action alternatives, the current value function V_n has N nodes, and worst case space expansion for regression and all Apply operations, the overall size of the result and the time complexity for one iteration are $O(C^{M^2(N+1)})$. However note that this is the worst case analysis and does not take reductions into account. While our method is not guaranteed to always work efficiently, the alternative of grounding the MDP will have an unmanageable number of states to deal with. Therefore, despite the high worst case complexity our method provides a potential improvement. As the next example illustrates reductions can substantially decrease diagram size and therefore save considerable time in computation.

3.8.7 A Comprehensive Example of Value Iteration

Figure 3.13 traces steps in the application of value iteration to the logistics domain. The TVDs, action choice probabilities, and reward function for this domain are given in Figure 3.8. To simplify the presentation, we continue using the predicate ordering $Bin \prec$ "=" $\prec On \prec Tin \prec rain$ introduced earlier.[2]

Given $V_0 = R$ as shown in Figure 3.13(a), Figure 3.13(b) gives the result of regression of V_0 through the action alternative $unloadS(b^*, t^*, c^*)$ by block replacement, denoted as $Regr(V_0, unloadS(b^*, t^*, c^*))$.

Figure 3.13(c) gives the result of multiplying $Regr(V_0, unloadS(b^*, t^*, c^*))$ with the choice probability of $unloadS$ $Pr(unloadS(b^*, t^*, c^*))$.

Figure 3.13(d) is the result of $Pr(unloadF(b^*, t^*, c^*)) \otimes Regr(V_0, unloadF(b^*, t^*, c^*))$. Notice that this diagram is simpler since $unloadF$ does not change the state and the TVDs for it are trivial.

Figure 3.13(e) gives the unreduced result of adding two outcomes for $unload(b^*, t^*, c^*)$, i.e.,
$[Pr(unloadS(b^*, t^*, c^*)) \otimes Regr(V_0,$ $unloadS(b^*, t^*, c^*))] \oplus [Pr(unloadF(b^*, t^*, c^*)) \otimes Regr(V_0,$ $unloadF(b^*, t^*, c^*))]$. Note that we first standardize apart the diagrams for action alternatives $unloadS(b^*, t^*, c^*)$ and $unloadF(b^*, t^*, c^*)$ by respectively renaming b as b_1 and b_2. Action parameters b^*, t^* and c^* at this stage are considered as constants and we do not change them. Also note that the recursive part of Apply (addition $\oplus$) has performed some reductions, i.e., removing the node $rain$ when both of its children lead to value 10.

In Figure 3.13(e), we can apply R7 to node $Bin(b_2, Paris)$ in the left branch. The conditions
P7.1: $[\exists b_1, Bin(b_1, Paris)] \rightarrow [\exists b_1, b_2, Bin(b_1, Paris) \wedge Bin(b_2, Paris)]$,
V7.1: $min(Bin(b_2, Paris)_{\downarrow t}) = 10 \geq max(Bin(b_2, Paris)_{\downarrow f}) = 9$,

[2] The details do not change substantially if we use the order suggested in Section3.5 (where equality is first).

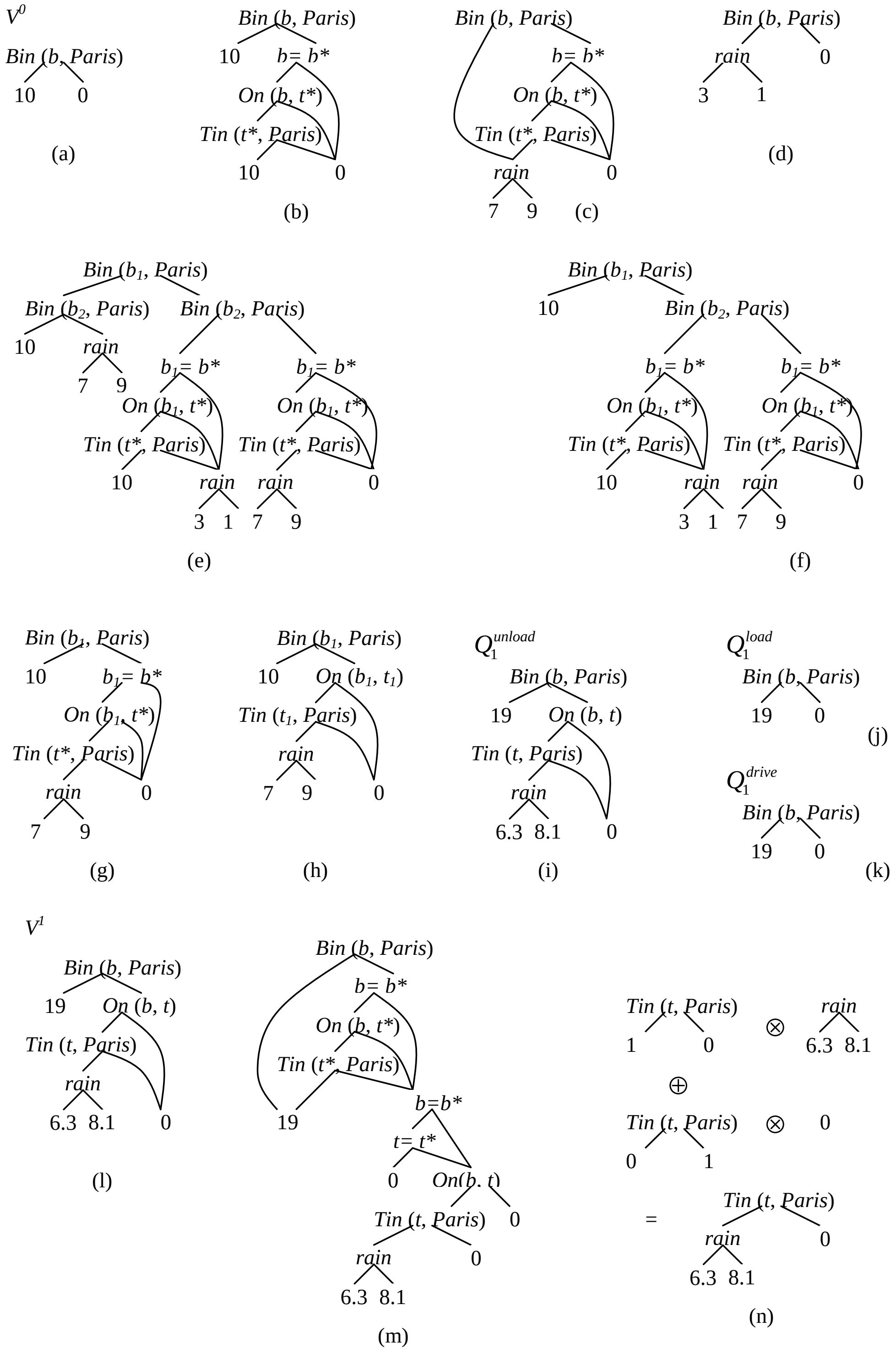


Figure 3.13: An example of value iteration in the Logistics Domain.

V7.2: $Bin(b_2, Paris)_{\downarrow t}$ is a constant
hold. According to Lemma 3 and Lemma 5 we can drop node $Bin(b_2, Paris)$ and connect its parent $Bin(b_1, Paris)$ to its true branch. Figure 3.13(f) gives the result after this reduction.

Next, consider the `true` child of $Bin(b_2, Paris)$ and the `true` child of the root. Conditions
P7.1: $[\exists b_1, b_2, \neg Bin(b_1, Paris) \wedge Bin(b_2, Paris)] \rightarrow [\exists b_1, Bin(b_1, Paris)]$,
V7.1: $min(Bin(b_1, Paris)_{\downarrow t}) = 10 \geq max(Bin(b_2, Paris)_{\downarrow t}) = 10$,
V7.2: $min(Bin(b_1, Paris)_{\downarrow t}) = 10 \geq max(Bin(b_2, Paris)_{\downarrow f}) = 9$
hold. According to Lemma 3 and Lemma 5, we can drop the node $Bin(b_2, Paris)$ and connect its parent $Bin(b_1, Paris)$ to $Bin(b_2, Paris)_{\downarrow f}$. Figure 3.13(g) gives the result after this reduction and now we get a fully reduced diagram. This is $T_{V_0}^{unload(b^*,t^*,c^*)}$.

In the next step we perform object maximization to maximize over action parameters b^*, t^* and c^* and get the best instance of the action $unload$. Note that b^*, t^* and c^* have now become variables, and we can perform one more reduction: we can drop the equality on the right branch by R9. Figure 3.13(h) gives the result after object maximization, i.e., obj-max$(T_{V_0}^{unload(b^*,t^*,c^*)})$. Note that we have renamed the action parameters to avoid the repetition between iterations.

Figure 3.13(i) gives the reduced result of multiplying the diagram in Figure 3.13(h), obj-max $(T_{V_0}^{unload(b^*,t^*,c^*)})$, by $\gamma = 0.9$, and adding the reward function. This result is Q_1^{unload}.

We can calculate Q_1^{load} and Q_1^{drive} in the same way and results are shown in Figure 3.13(j) and Figure 3.13(k) respectively. For $drive$ the TVDs are trivial and the calculation is relatively simple. For $load$, the potential loading of a box already in Paris is dropped from the diagram by the reduction operators in the process of object maximization.

Figure 3.13(l) gives V_1, the result after maximizing over Q_1^{unload}, Q_1^{load} and Q_1^{drive}. Here again we standardized apart the diagrams, maximized over them, and then reduced the result. In this case the diagram for $unload$ dominates the other actions. Therefore Q_1^{unload} becomes V_1, the value function after the first iteration.

Now we can start the second iteration, i.e., computing V_2 from V_1. Figure 3.13(m) gives the result of block replacement in the regression of V^1 through the action alternative $unloadS(b^*, t^*, c^*)$. Note that we have sorted the TVD for $on(B, T)$ so that it obeys the ordering we have chosen. However, the diagram resulting from block replacement is not sorted.

To address this we use the block combination algorithm to combine blocks bottom up. Figure 3.13(n) illustrates how we combine blocks $Tin(t, Paris)$, which is a TVD, and its two children, which have been processed and are general FODDs. After we combine $Tin(t, Paris)$ and its two children, $On(b, t)_{\downarrow t}$ has been processed. Since $On(b, t)_{\downarrow f} = 0$, now we can combine $On(b, t)$ and its two children in the next step of block combination. Continuing this process we get a sorted representation of $Regr(V_1, unloadS(b^*, t^*, c^*))$.

3.8.8 Extracting Optimal Policies

There is more than one way to represent policies with FODDs. Here we simply note that a policy can be represented implicitly by a set of regressed value functions. After the value iteration terminates, we can perform one more iteration and compute the set of Q-functions using Equation 3.1.

Then, given a state s, we can compute the maximizing action as follows:

1. For each Q-function $Q^{A(\vec{x})}$, compute $\text{MAP}_{Q^{A(\vec{x})}}(s)$, where $\vec{x}$ are considered as variables.

2. For the maximum map obtained, record the action name and action parameters (from the valuation) to obtain the maximizing action.

This clearly implements the policy represented by the value function. We use this approach in our implementation. An alternative approach that represents the policy explicitly was developed in the context of a policy iteration algorithm (Wang & Khardon, 2007).

3.9 Summary and Concluding Remarks

In this chapter we presented the definition of FODDs along with their properties and algorithms for combining and reducing them. In particular we developed the reduction operator R7 and discussed its applicability. We also presented an SDP based VI algorithm for RMDPs that uses FODDs as the underlying RMDP language. In subsequent chapters we will build on this framework, introduce improvements and provide experimental evaluation of these ideas.

Chapter 4

Theorem Proving Reductions

The VI algorithm we presented in the last chapter performs reasoning with FODDs. This reasoning process introduces many redundancies in the FODDs. In the previous chapter we also presented a set of operators to reduce FODDs. Reductions are based on the idea that a diagram can be made smaller by the removal of these redundancies. This set of reduction operators, however, is by no means comprehensive and has two limitations. Firstly it falls short of providing a canonical form for FODDs. Secondly there are simple cases where FODDs can be simplified but none of the existing reductions (R1 $\cdots$ R5, R7) can perform the simplification. Unfortunately an implementation of the VI algorithm using this set of reduction operators is too slow to yield practical results because diagrams are not sufficiently reduced and because the reductions themselves are expensive.

In this chapter, we address the second issue by introducing new reduction operators R9, R10 and R11. R9 is designed specifically to reduce diagrams with redundant equality nodes. R10 is similar to R7 in some sense. Unlike R7, however, R10 makes a global analysis of the FODD and removes many redundant portions of the diagram simultaneously. R11 works locally and targets a particular redundancy that arises quite often when FODDs are composed. The new operators still do not guarantee a canonical form, but improve upon many of the deficiencies of the previous operators. All of these reductions rely on proving reachability conditions (e.g. P7.2) and thus make use of theorem proving. We therefore call them theorem proving reductions to distinguish them from the constructions in chapter 6. In addition this chapter includes results that directly improve the previous

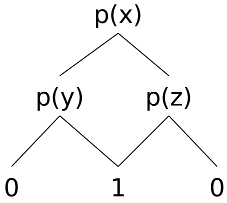


Figure 4.1: Example illustrating the need for a DPO

constructions. First we introduce a technique to improve applicability of R7 and secondly we show that standardizing apart diagrams is unnecessary when combining FODDs under the max operator.

We need the following notation in addition to that explained in the previous chapter. If B is a FODD and p is a path from the root to a leaf in B, then the path formula for p, denoted by PF(p) is the conjunction of literals along p. The variables of p, are denoted $\vec{x^p}$. When $\vec{x^p}$ are existentially quantified, satisfiability of PF(p) under an interpretation I is a necessary and sufficient condition for the path p to be traversed by some valuation under I. If ζ is such a valuation, then we define $Path_B(I, \zeta) = p$. The leaf reached by path p is denoted as $leaf(p)$. We let PF(p)$\backslash Lit$ denote the path formula of path p with the literal Lit removed (if it was present) from the conjunction.

The following definitions are important in developing new reductions and to understand potential scope for reducing diagrams.

Definition 3 *A descending path ordering (DPO) is an ordered list of all paths from the root to a leaf in FODD B, sorted in descending order by the value of the leaf reached by the path. The relative order of paths reaching the same leaf is unimportant as long as it is fixed.*

Definition 4 *If B is a FODD, and P is the DPO for B, then a path $p_j \in P$ is instrumental iff*

1. *there is an interpretation I and valuation, ζ, such that $Path_B(I, \zeta) = p_j$, and*

2. *$\forall$ valuations η, if $Path_B(I, \eta) = p_k$, then $k \geq j$.*

The example in Figure 4.1 shows why a DPO is needed. The paths $p(x) \wedge \neg p(y)$ and $\neg p(x) \wedge p(z)$ both imply each other. Whenever there is a valuation traversing one of the paths there is always another valuation traversing the other. Removing any one path from the diagram would be safe meaning that the map is not changed. But we cannot remove both paths. Without an externally imposed order on the paths, it is not clear which path should be labeled as redundant. A DPO does exactly that to make the reduction possible.

4.1 (R9) Equality Reduction

Consider a FODD B with an equality node n labeled $t = x$. Sometimes we can drop n and connect its parents to a sub-FODD that is the result of taking the maximum of the left and the right children of n. For this reduction to be applicable B has to satisfy the following condition.

(E9.1) : For an equality node n labeled $t = x$ at least one of t and x is a variable and it appears neither in $n_{\downarrow f}$ nor in the node formula for n. To simplify the description of the reduction procedure below, we assume that x is that variable.

Additionally we make the following assumption about the domain.

(D9.1) : The domain contains more than one object.

The above assumption guarantees that valuations reaching the right child of equality nodes exist. This fact is needed in proving correctness of the Equality reduction operator. First we describe the reduction procedure for R9(n). Let B_n denote the FODD rooted at node n in FODD B. We extract a copy of $B_{n_{\downarrow t}}$ (and name it $B_{n_{\downarrow t}}$-copy), and a copy of $B_{n_{\downarrow f}}$ ($B_{n_{\downarrow f}}$-copy) from B. In $B_{n_{\downarrow t}}$-copy, we rename the variable x to t to produce diagram $B'_{n_{\downarrow t}}$-copy. Let $B'_n = Apply(B'_{n_{\downarrow t}}\text{-copy}, B_{n_{\downarrow f}}\text{-copy}, max)$. Finally we drop the node n in B and connect its parents to the root of B'_n to obtain the final result B'. An example is shown in Figure 4.2.

Informally, we are extracting the parts of the FODD rooted at node n, one where $x = t$ (and renaming x to t in that part) and one where $x \neq t$. The condition E9.1 and the assumption D9.1 guarantee that regardless of the value of t, we have valuations reaching both parts. Since by the definition of map, we maximize over the valuations, in this case we can maximize over the diagram structure itself. We do this by calculating the function

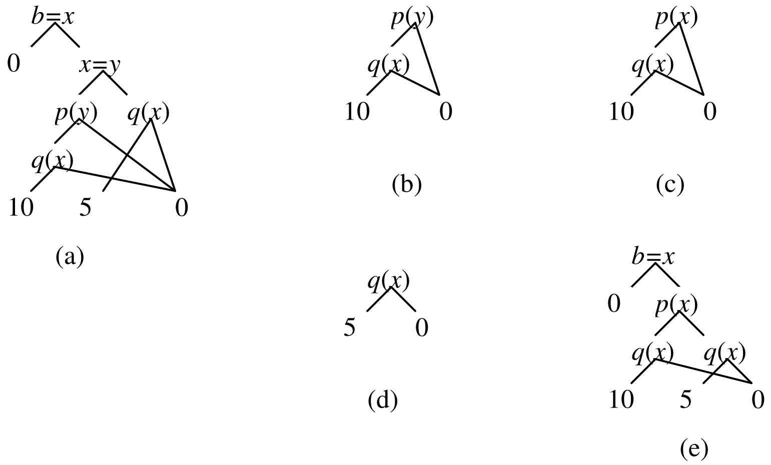


Figure 4.2: An example of the equality reduction. (a) The FODD before reduction. The node $x = y$ satisfies condition E9.1 for variable y. (b) $B_{n_{\downarrow t}}$-copy ($n_{\downarrow t}$ extracted). (c) $B_{n_{\downarrow t}}$-copy renamed to produce $B'_{n_{\downarrow t}}$-copy. (d) $B_{n_{\downarrow f}}$-copy. (e) Final result with node n replaced by $apply(B'_{n_{\downarrow t}}\text{-copy}, B_{n_{\downarrow f}}\text{-copy}, max)$

which is the maximum of the two functions corresponding to the two children of n (using $Apply$) and replacing the old sub-diagram rooted at node n by the new combined diagram. Theorem 12 proves that this does not affect the map of B.

One concern for implementation is that we simply replace the old sub-diagram by the new sub-diagram, which may result in a diagram where strong reductions are applicable. While this is not a problem semantically, we can avoid the need for strong reductions by using $Apply$ that implicitly performs strong reductions R1(neglect) and R2(join) as follows.

Let B_a denote the FODD resulting from replacing node n in B with 0, and B_b the FODD resulting from replacing node n with 1 and all leaves other than node n by 0, we have the final result $B' = B_a \oplus B'_b$ where $B'_b = B_b \otimes B'_n$. By correctness of $Apply$ the two forms of calculating B' give the same map.

In the following we prove that for any node n where equality condition E9.1 holds in B we can perform the equality reduction R9 without changing the map for any interpretation satisfying D9.1. We start with properties of FODDs defined above, e.g., B_a, B_b, and B'_b. Let Γ_n denote the set of all valuations reaching node n and let Γ_m denote the set of all valuations not reaching node n in B. From the basic definition of map we have the following:

Claim 1 *For any interpretation I,*
(a) $\forall\, \zeta \in \Gamma_m$, *MAP*$_{B_a}(I, \zeta) =$ *MAP*$_B(I, \zeta)$.
(b) $\forall\, \zeta \in \Gamma_n$, *MAP*$_{B_a}(I, \zeta) = 0$.
(c) $\forall\, \zeta \in \Gamma_m$, *MAP*$_{B_b}(I, \zeta) = 0$.
(d) $\forall\, \zeta \in \Gamma_n$, *MAP*$_{B_b}(I, \zeta) = 1$.

From Claim 1 and the definition of map, we have,

Claim 2 *For any interpretation I,*
(a) $\forall\, \zeta \in \Gamma_m$,*MAP*$_{B'_b}(I, \zeta) = 0$.
(b) $\forall\, \zeta \in \Gamma_n$,*MAP*$_{B'_b}(I, \zeta) =$ *MAP*$_{B'_n}(I, \zeta)$.

From Claim 1, Claim 2, and the definition of map we have,

Claim 3 *For any interpretation I,*
(a) $\forall\, \zeta \in \Gamma_m$, *MAP*$_{B'}(I, \zeta) =$ *MAP*$_B(I, \zeta)$.
(b) $\forall\, \zeta \in \Gamma_n$, *MAP*$_{B'}(I, \zeta) =$ *MAP*$_{B'_n}(I, \zeta)$.

Next we prove the main property of this reduction stating that for all valuations reaching node n in B, the old sub-FODD rooted at n and the new (combined) sub-FODD produce the same map.

Lemma 11 *Let Γ_n be the set of valuations reaching node n in FODD B. For any interpretation I satisfying D9.1,* $\max_{\zeta \in \Gamma_n}$ *MAP*$_{B_n}(I, \zeta) = \max_{\zeta \in \Gamma_n}$ *MAP*$_{B'_n}(I, \zeta)$.

Proof: By condition E9.1, the variable x does not appear in $NF(n)$ and hence its value in $\zeta \in \Gamma_n$ is not constrained. We can therefore partition the valuations in Γ_n into disjoint sets, $\Gamma_n = \{\Gamma_\Delta \mid \Delta$ is a valuation to variables other than $x\}$, where in Γ_Δ variables other than x are fixed to their value in Δ and x can take any value in the domain of I. Assumption D9.1 guarantees that every Γ_Δ contains at least one valuation reaching $B_{n_{\downarrow t}}$ and at least one valuation reaching $B_{n_{\downarrow f}}$ in B. Note that if a valuation ζ reaches $B_{n_{\downarrow t}}$ then $t = x$ is satisfied by ζ thus $\text{MAP}_{B_{n_{\downarrow t}}}(I, \zeta) = \text{MAP}_{B'_{n_{\downarrow t}}}$-copy$(I, \zeta)$. Since x does not appear in $B_{n_{\downarrow f}}$ we also have that MAP$_{B'_{n_{\downarrow f}}}$-copy(I, ζ) is constant for all $\zeta \in \Gamma_\Delta$. Therefore by the correctness of $Apply$ we have $\max_{\zeta\in\Gamma_\Delta}$ MAP$_{B_n}(I, \zeta) = \max_{\zeta\in\Gamma_\Delta}$ MAP$_{B'_n}(I, \zeta)$.

Finally, by the definition of map,

$$\begin{aligned} \max_{\zeta\in\Gamma_n} \text{MAP}_{B_n}(I,\zeta) &= \max_{\Delta} \max_{\zeta\in\Gamma_\Delta} \text{MAP}_{B_n}(I,\zeta) \\ &= \max_{\Delta} \max_{\zeta\in\Gamma_\Delta} \text{MAP}_{B'_n}(I,\zeta) \\ &= \max_{\zeta\in\Gamma_n} \text{MAP}_{B'_n}(I,\zeta). \end{aligned}$$

■

Lemma 12 *Let B be a FODD, n a node for which condition E9.1 holds, and B' be the result of R9(n), then for any interpretation I satisfying D9.1, MAP$_B(I)$ = MAP$_{B'}(I)$.*

Proof: Let $X = \max_{\zeta\in\Gamma_m}$ MAP$_{B'}(I, \zeta)$ and $Y = \max_{\zeta\in\Gamma_n}$ MAP$_{B'}(I, \zeta)$. By the definition of map, MAP$_{B'}(I) = max(X, Y)$. However, by Claim 3, $X = \max_{\zeta\in\Gamma_m}$ MAP$_B(I, \zeta)$ and by Claim 3 and Lemma 11, $Y = \max_{\zeta\in\Gamma_n}$ MAP$_{B'_n}(I, \zeta) = \max_{\zeta\in\Gamma_n}$ MAP$_{B_n}(I, \zeta)$. Thus $max(X, Y)$ = MAP$_B(I)$ = MAP$_{B'}(I)$. ■

While Lemma 12 guarantees correctness, when applying it in practice it may be important to avoid violations of the sorting order (which would require expensive re-sorting of the diagram). If both x and t are variables, we can sometimes replace both with a new variable name so the resulting diagram is sorted. However this is not always possible. When such a violation is unavoidable, there is a trade-off between performing the reduction and sorting the diagram and ignoring the potential reduction. In our implementation we always avoid sorting the diagram and choose to ignore the reduction if it causes a sorting violation.

4.2 The R10 Reduction

A path in FODD B is dominated if whenever a valuation traverses it, there is always another valuation traversing another path and reaching a leaf of greater or equal value. Now if all paths through an edge e are dominated, then no valuation crossing that edge will ever determine the map under max aggregation semantics. In such cases we can replace target(e) by a 0 leaf. This is the basic intuition behind the R10 operation which is similar to the reduction of rules in decision lists of Kersting et al. (2004).

Although its objective is the same as that of R7-replace, R10 is faster to compute and has two advantages over R7-replace. First, because paths can be ranked by the value of the leaf they reach, we can perform a single ranking and check for all dominated paths (and hence all dominated edges). Hence, while all other reduction operators are local, R10 is a global reduction. Second, the theorem proving required for R10 is always on conjunctive formulas with existentially quantified variables. This gives a speedup over R7-replace.

Consider the example shown in Figure 4.3. The following is a DPO for this diagram:

1. $p(y)$, $\neg p(z)$, $\neg p(x) \rightarrow 3$
2. $p(y)$, $\neg p(z)$, $p(x)$, $\neg q(x) \rightarrow 3$
3. $\neg p(y)$, $p(x)$, $q(x) \rightarrow 2$
4. $p(y)$, $\neg p(z)$, $p(x)$, $q(x) \rightarrow 2$

Notice that the relative order of paths reaching the same leaf in this DPO is defined by ranking shorter paths higher than longer ones. This is not a requirement for the correctness of the algorithm but is a good heuristic. According to the reduction procedure, all edges of path 1 are important and cannot be reduced. However, since 1 subsumes 2, 3 and 4, all the other edges (those belonging to paths 2, 3 and 4 and those not appearing in any of the ranked paths) can be reduced. Therefore the reduction procedure replaces the targets of all edges other than the ones in path 1, to the value 0. Path 1 is thus an *instrumental* path but paths 2, 3 and 4 are not. We now present a formal definition of R10.

Procedure 3 *R10(B)*

1. *Let E be the set of all edges in B*
2. *Let $P = [p_1, p_2 \cdots p_n]$ be a DPO for B. Thus p_1 is a path reaching the highest leaf and p_n is a path reaching the lowest leaf.*
3. *For $j = 1$ to n, do the following*

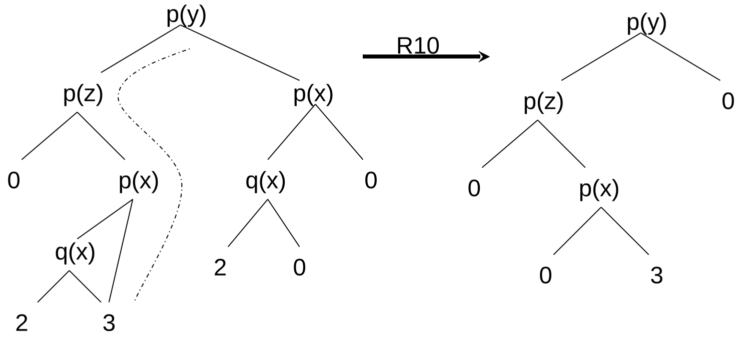


Figure 4.3: Example of R10 reduction

   (a) Let E_{p_j} be the set of edges on p_j

   (b) If $\neg\exists i$, $i < j$ such that $\mathcal{B} \models (\exists \vec{x^{p_j}}$, PF($p_j$)) $\rightarrow$ ($\exists \vec{x^{p_i}}$, PF(p_i)), then set $E = E - E_{p_j}$

4. For every edge $e \in E$, set target(e) $= 0$ in B

In the example in Figure 4.3 none of the paths 2, 3 and 4 satisfy the conditions of step 3b in the algorithm. Therefore their edges are not to be removed from E and are assigned the value 0 by the algorithm. Here R10 is able to identify in one pass, the one path (shown along a curved indicator line) that dominates all other paths. To achieve the same reduction, R7-replace takes 2-3 passes depending on the order of application. Since every pass of R7-replace has to check for implication of edge formulas for every pair of edges, this can be expensive. On the other hand, there are cases where R10 is not applicable but R7-replace is. An example of this is shown in the diagram in Figure 4.4. For this diagram it is easy to see that if e_2 is reached then so is e_1 and e_1 always gives a strictly better value. R10 cannot be applied because it tests subsumption for complete paths. In this case the path for e_2 implies the disjunction of two paths going through e_1. We now present a proof of correctness for R10.

Lemma 13 *For any path $p_j \in P$, if p_j is instrumental then $\neg\exists i$, $i < j$ and $\mathcal{B} \models (\exists \vec{x^{p_j}}$, PF($p_j$)) $\rightarrow$ ($\exists \vec{x^{p_i}}$, PF(p_i))*

Proof: If p_j is instrumental then by definition, there is an interpretation I and valuation, ζ, such that $Path_B(I, \zeta) = p_j$, and $\forall$ valuations η, $\neg\exists\ i < j$ such that $Path_B(I, \eta) = p_i$. In

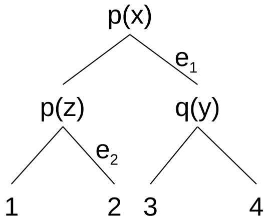


Figure 4.4: Example where R7 is applicable but R10 is not

other words, $I \models [\mathcal{B} \rightarrow (\exists x^{\vec{p_j}}$, PF($p_j$))] but $I \not\models [\mathcal{B} \rightarrow (\exists x^{\vec{p_i}}$, PF($p_i$))] for any $i < j$. This implies that $\neg\exists i$, $i < j$ and ($\mathcal{B} \rightarrow \exists x^{\vec{p_j}}$, PF($p_j$)) $\models$ ($\mathcal{B} \rightarrow \exists x^{\vec{p_i}}$, PF($p_i$)). Hence $\neg\exists i$, $i < j$ and $\mathcal{B} \models [(\exists x^{\vec{p_j}}$, PF($p_j$)) $\rightarrow$ ($\exists x^{\vec{p_i}}$, PF(p_i))]. ■

The above lemma basically states that the conjunctive formula for an instrumental path cannot logically imply the conjunctive formula of any other preferred path (with respect to the DPO) in the FODD.

Lemma 14 *If E is the set of edges left at the end of the R10 procedure then if $e \in E$ then there is no instrumental path that goes through e.*

Proof: Lemma 13 proves that if a path p_j is instrumental, then $\neg\exists i$, $i < j$ and $\mathcal{B} \models [(\exists x^{\vec{p_j}}$, PF($p_j$)) $\rightarrow$ ($\exists x^{\vec{p_i}}$, PF(p_i))]. Thus in step 3b of R10, if a path is instrumental, all its edges are removed from E. Therefore if $e \in E$ at the end of the R10 procedure, it cannot be in p_j. Since p_j is not constrained in any way in the argument above, e cannot be in any instrumental path. ■

Given the above 2 lemmas, the following theorem makes the argument that R10 preserves instrumental paths by preserving the edges on instrumental paths and thus maintains the map of any interpretation.

Theorem 4 *Let B be any FODD. If $B' =$ R10(B) then $\forall$ interpretations I, MAP$_B(I) =$ MAP$_{B'}(I)$*

Proof: By the definition of R10, the only difference between B and B' is that some edges that pointed to sub-FODDs in B, point to the 0 leaf in B'. These are the edges left in the set E at the end of the R10 procedure. Therefore any valuation crossing these

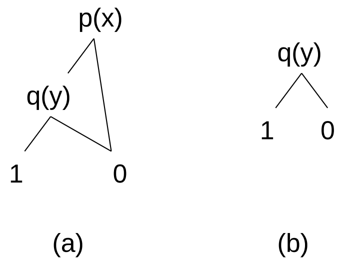


Figure 4.5: Example of R11 reduction

edges achieves a value of 0 in B' but could have achieved more value in B under the same interpretation. Valuations not crossing these edges will achieve the same value in B' as they did in B. Therefore for any interpretation I and valuation ζ, MAP$_B(I, \zeta) \geq$ MAP$_{B'}(I, \zeta)$ and hence MAP$_B(I) \geq$ MAP$_{B'}(I)$.

Fix any interpretation I and $v =$ MAP$_B(I)$. Let ζ be a valuation such that MAP$_B(I, \zeta)$ $= v$. If there is more than one ζ that gives value v, we choose one whose path p_j has the least index in P. Now by definition p_j is instrumental and by lemma 14, none of the edges of p_j are removed by R10. Therefore MAP$_{B'}(I, \zeta) = v =$ MAP$_B(I)$. By the definition of the max aggregation semantics, MAP$_{B'}(I) \geq$ MAP$_{B'}(I, \zeta)$ and therefore MAP$_{B'}(I) \geq$ MAP$_B(I)$ ■

4.3 The R11 Reduction

Consider the FODD B in Figure 4.5(a). Clearly, with no background knowledge this diagram cannot be reduced. Now assume that the background knowledge $\mathcal{B}$ contains a rule $\forall x, [q(x) \rightarrow p(x)]$. In this case if there exists a valuation that reaches the 1 leaf, there must be another such valuation ζ that agrees on the values of x and y. ζ dominates the other valuations under the max aggregation semantics. The background knowledge rule implies that for ζ, the test at the root node is redundant. However, we cannot set the left child of the root to 0 since the entire diagram will be eliminated. Therefore R7 is not applicable, and similarly none of the other existing reductions is applicable. Although the example is artificial, similar situations arise often in runs of the value iteration algorithm.[1] We introduce the R11 reduction operator that can handle such situations. R11 reduces the FODD in Figure 4.5(a) to the FODD in Figure 4.5(b)

Let B be a FODD, n a node in B, e an edge such that $e \in \{n_{\downarrow t}, n_{\downarrow f}\}$, $e' =$ sibling(e) (so that when $e = n_{\downarrow t}$, $e' = n_{\downarrow f}$ and vice versa), and P the set of all paths from the root to a non-zero leaf going through edge e. Then the reduction $R11(B, n, e)$ drops node n from diagram B and connects its parents to target(e). We need two conditions for the applicability of R11. The first requires that the sibling is a zero valued leaf.

Condition 1 target(e') $= 0$

The second requires that valuations that are rerouted by R11 when traversing B' are dominated by other valuations giving the same value.

Condition 2 $\forall p \in P$, $\mathcal{B} \models [\exists \vec{x^p}$, PF($p$)$\backslash n^e.lit \wedge n^{e'}.lit] \rightarrow [\exists \vec{x^p}$, PF($p$)]

Theorem 5 *If $B' = R11(B, n, e)$, and conditions 1 and 2 hold, then $\forall$ interpretations I, MAP$_B(I) =$ MAP$_{B'}(I)$*

Proof: Let I be any interpretation and let Z be the set of all valuations. We can divide Z into three disjoint sets depending on the path taken by valuations in B under I. Z^e - the set of all valuations crossing edge e, $Z^{e'}$ - the set of all valuations crossing edge e' and Z^{other} - the set of valuations not reaching node n. We analyze the behavior of the valuations in these sets under I.

- Since structurally the only difference between B and B' is that in B' node n is bypassed, all paths from the root to a leaf that do not cross node n remain untouched. Therefore $\forall \zeta \in Z^{other}$, MAP$_B(I, \zeta) =$ MAP$_{B'}(I, \zeta)$.

[1] This happens naturally, without the artificial background knowledge used for our example. The main reason is that standardizing apart introduces multiple renamed copies of the same atoms in the different diagrams. When the diagrams are added, many of the atoms are redundant but some are not removed by old operators. These atoms may end up in a parent-child relation with weak implication from child to parent, similar to the example given.

- Since, in B' the parents of node n are connected to target(e), all valuations crossing edge e and reaching target(e) in B under I will be unaffected in B' and will, therefore, produce the same map. Thus $\forall \zeta \in Z^e$, MAP$_B(I, \zeta) =$ MAP$_{B'}(I, \zeta)$.

- Now, let m denote the node target(e) in B. Under I, all valuations in $Z^{e'}$ will reach the 0 leaf in B but they will cross node m in B'. Depending on the leaf reached after crossing node m, the set $Z^{e'}$ can be further divided into 2 disjoint subsets. $Z^{e'}_{zero}$ - the set of valuations reaching a 0 leaf and $Z^{e'}_{nonzero}$ - the set of valuations reaching a non-zero leaf. Clearly $\forall \zeta \in Z^{e'}_{zero}$, MAP$_B(I, \zeta) =$ MAP$_{B'}(I, \zeta)$.

  By the structure of B, every $\zeta \in Z^{e'}_{nonzero}$, traverses some $p \in P$, that is, (PF(p)$\backslash n^e.lit$ $\wedge$ $n^{e'}.lit$)ζ is true in I. Condition 2 states that for every such ζ, there is another valuation η such that (PF(p))η is true in I, so η traverses the same path. However, every such valuation η must belong to the set Z^e by the definition of the set Z^e. In other words, in B' every valuation in $Z^{e'}_{nonzero}$ is dominated by some valuation in Z^e.

From the above argument we conclude that in B' under I, every valuation either produces the same map as in B or is dominated by some other valuation. Under the max aggregation semantics, therefore, MAP$_B(I) =$ MAP$_{B'}(I)$. ∎

4.4 Further Speedup of Theorem Proving Reductions

Identifying and applying more reductions helps keep FODDs compact. This improves efficiency of the combination and reasoning procedures. This Section presents two techniques to further speed up theorem proving reductions while maintaining an exact solution.

4.4.1 Subtracting Apart - Improving applicability of R7

Consider the FODD B in Figure 4.6. Intuitively a weak reduction is applicable on this diagram because of the following argument. Consider a valuation $\zeta = \{x \setminus 1, y \setminus 2, z \setminus 3\}$ crossing edge e_2 under some interpretation I. $I \models \mathcal{B} \rightarrow \neg p(1) \wedge p(2)$. Therefore there must be a valuation $\eta = \{x \setminus 2, z \setminus 3\}$ (and any value for y), that crosses edge e_1. Now depending on the truth value of $I \models \mathcal{B} \rightarrow q(1)$ and $I \models \mathcal{B} \rightarrow q(2)$, we have four possibilities of

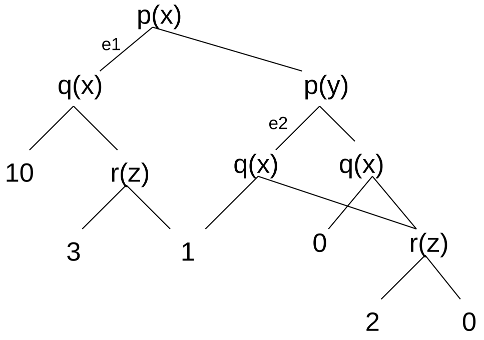


Figure 4.6: Sub-Apart

where ζ and η would reach after crossing the nodes target(e_2) and target(e_1) respectively. However, in all these cases, $\text{MAP}_B(I, \eta) \geq \text{MAP}_B(I, \zeta)$. Therefore we should be able to replace target(e_2) by a 0 leaf. A similar argument shows that we should also be able to drop the node source(e_2). Surprisingly, though, none of the R7 conditions apply in this case and this diagram cannot be reduced. On closer inspection we find that the reason for this is that the conditions **(P7.2)** and **(V7.3)** are too restrictive. **(V7.3)** holds but **(P7.2)** requires that $\forall x, \forall z$,$[[\exists y, \neg p(x) \wedge p(y)] \rightarrow [p(x)]]$ implying that for every valuation crossing edge e_2, there has to be another valuation crossing edge e_1 such that the valuations agree on the value of x and z and this does not hold. However, from our argument above, for η to dominate ζ, the two valuations need not agree on the value of x. We observe that if we rename variable x so that its instances are different in the sub-FODDs rooted at target(e_1) and target(e_2) (i.e. we standardized apart w.r.t. x) then both **(P7.2)** and **(V7.3)** go through and the diagram can be reduced. To address this problem, we introduce a new FODD subtraction algorithm *sub-apart*. Given diagrams B_1 and B_2 the algorithm tries to standardize apart as many of their common variables as possible, while keeping the condition $B_1 \ominus B_2 \geq 0$ true. The algorithm returns a 2-tuple $\{T, V\}$, where T is a boolean variable indicating whether the combination can produce a diagram that has no negative leaves when all variables except the ones in V are standardized apart.

Procedure 4 *sub-apart(A, B)*

1. *If A and B are both leaves,*

```
    (a) If $A - B \geq 0$ return $\{true, \{\}\}$ else return $\{false, \{\}\}$
2. If $l(A) \prec l(B)$, let
    (a) $\{L, V_1\}$ = sub-apart(target($A_{\downarrow t}$), $B$)
    (b) $\{R, V_2\}$ = sub-apart(target($A_{\downarrow f}$), $B$)
   Return $\{L \wedge R, V_1 \cup V_2\}$
3. If $l(B) \prec l(A)$, let
    (a) $\{L, V_1\}$ = sub-apart($A$, target($B_{\downarrow t}$))
    (b) $\{R, V_2\}$ = sub-apart($A$, target($B_{\downarrow f}$))
   Return $\{L \wedge R, V_1 \cup V_2\}$
4. If $l(A) = l(B)$, let $V$ be the variables of $A$ (or $B$). Let
    (a) $\{LL, V_3\}$ = sub-apart(target($A_{\downarrow t}$), target($B_{\downarrow t}$))
    (b) $\{RR, V_4\}$ = sub-apart(target($A_{\downarrow f}$), target($B_{\downarrow f}$))
    (c) $\{LR, V_5\}$ = sub-apart(target($A_{\downarrow t}$), target($B_{\downarrow f}$))
    (d) $\{RL, V_6\}$ = sub-apart(target($A_{\downarrow f}$), target($B_{\downarrow t}$)
    (e) If $LL \wedge RR = false$, return $\{false, V_3 \cup V_4\}$
    (f) If $LR \wedge RL = false$ return $\{true, V \cup V_3 \cup V_4\}$
    (g) Return $\{true, V_3 \cup V_4 \cup V_5 \cup V_6\}$
```

The next theorem shows that the procedure is correct. The variables common to B_1 and B_2 are denoted by $\vec{u}$ and $B^{\vec{w}}$ denotes the combination diagram of B_1 and B_2 under the subtract operation when all variables except the ones in $\vec{w}$ are standardized apart. Let n_1 and n_2 be the roots nodes of B_1 and B_2 respectively.

Theorem 6 *sub-apart(n_1, n_2) = $\{true, \vec{v}\}$ implies $B^{\vec{v}}$ contains no negative leaves and sub-apart(n_1, n_2) = $\{false, \vec{v}\}$ implies $\neg\exists\vec{w}$ such that $\vec{w} \subseteq \vec{v}$ and $B^{\vec{w}}$ contains no negative leaves.*

Proof: The proof is by induction on k, the sum of the number of nodes in B_1 and B_2. For the base case when $k = 2$, both B_1 and B_2 are single leaf diagrams and the statement is trivially true. Assume that the statement is true for all $k \leq m$ and consider the case where $k = m + 1$. When $l(n_1) \prec l(n_2)$, in the resultant diagram of combination under subtraction, we expect n_1 to be the root node and $n_{1\downarrow t} \ominus n_2$ and $n_{1\downarrow f} \ominus n_2$ to be the left and right sub-FODDs respectively. Hence, the sub-apart algorithm recursively calls sub-apart($n_{1\downarrow t}$, n_2) and sub-apart($n_{1\downarrow f}$, n_2). Since the sum of the number of nodes of the diagrams in the recursive calls is always $\leq m$, the statement is true for both recursive calls. Clearly, the top level can return a `true` iff both calls return `true`. A similar argument shows that the statement is true when $l(n_2) \prec l(n_1)$.

When $l(n_1) = l(n_2)$, again by the inductive hypothesis, the statement of the theorem is true for all recursive calls. Here we have 2 choices. We could either standardize apart the variables V in $l(n_1)$ and $l(n_2)$ or keep them identical. If they are the same, in the resultant diagram of combination under subtraction we expect n_1 to be the root node and $n_{1\downarrow t} \ominus n_{2\downarrow t}$ and $n_{1\downarrow f} \ominus n_{2\downarrow f}$ to be the left and right sub-FODDs respectively. Again the top level can return a `true` iff both calls return `true` . The set of shared variables requires the variables of $l(n_1)$ in addition to those from the recursive calls in order to ensure that $l(n_1) = l(n_2)$.

If we standardize apart $l(n_1)$ and $l(n_2)$, then we fall back on one of the cases where $n_1 \neq n_2$ except that the algorithm checks for the second level of recursive calls $n_{1\downarrow t} \ominus n_{2\downarrow t}$, $n_{1\downarrow t} \ominus n_{2\downarrow f}$, $n_{1\downarrow f} \ominus n_{2\downarrow t}$ and $n_{1\downarrow f} \ominus n_{2\downarrow f}$. The top level of the algorithm can return `true` if all four calls return `true` and return the union of the sets of variables returned by the four calls. If not all four calls return `true`, the algorithm can still keep the variables in $l(n_1)$ and $l(n_2)$ identical and return `true` if the conditions for that case are met. ■

The theorem shows that the algorithm is correct but does not guarantee minimality. In fact, as we described in Chapter 3 (Figure 3.6), the smallest set of variables $\vec{w}$ for $B^{\vec{w}}$ to have no negative leaves may not be unique and one can also show that the output of sub-apart may not be minimal. In principle, one can use a greedy procedure that standardizes apart one variable at a time and arrives at a minimal set $\vec{w}$. However, although *sub-apart* does not produce a minimal set, we prefer it to the greedy approach because it is fast and often generates a small set $\vec{w}$ in practice. We can now define new conditions for applicability of R7:

(V7.3S) : sub-apart(target(e_1), target(e_2)) = $\{true, V_1\}$.

(P7.2S) : $\mathcal{B} \models \forall\ V_1$, $[[\exists \vec{w}, \text{EF}(e_2)] \rightarrow [\exists \vec{v}, \text{EF}(e_1)]]$ where V_1 is as above and $\vec{v}$, $\vec{w}$ are the remaining variables (i.e. not in V_1) in EF(e_1), EF(e_2) respectively.

(P7.2S) guarantees that whenever there is a valuation ζ_2 going through target(e_2), there is always a ζ_1 going through target(e_1) and ζ_1 and ζ_2 agree on V_1. **(V7.3S)** guarantees that under this condition, ζ_1 provides a better value than ζ_2. In fact the proof of Lemma 4 goes through unchanged. For completeness we include the proof here.

Lemma 15 *Let B be a FODD, e_1 and e_2 edges for which conditions P7.2S, V7.3S, and S1 hold, and B' the result of R7-replace(b, e_1, e_2), where $0 \leq b \leq \min(target(e_1))$, then for any interpretation I we have MAP$_B(I)$ = MAP$_{B'}(I)$.*

Proof: Consider any valuation ζ_2 that reaches $target(e_2)$. By P7.2S there is another valuation ζ_1 reaching $target(e_1)$ and ζ_2 and ζ_1 agree on all variables that appear in V_1. Therefore, by V7.3S ζ_1 achieves a higher value. Therefore according to maximum aggregation the value of MAP$_B(I)$ will never be determined by $target(e_2)$, and we can replace it with a constant as described above. ■

Importantly, conditions **(P7.2S)** , **(V7.3S)** subsume all the previous conditions for applicability and safety of R7-replace that were given in Chapter 3. I.e. whenever any of the two previous conditions for the applicability of R7-replace (P7.1, V7.1 and S1 or P7.2, V7.3 and S1) are satisfied, P7.2S, V7.3S and S1 are satisfied. A very similar argument shows how *sub-apart* extends and simplifies the conditions of R7-drop. Thus the use of *sub-apart* both simplifies the conditions to be tested and provides more opportunities for reductions. In our implementation, we use the new conditions with sub-apart (instead of the old conditions) whenever testing for applicability of R7.

4.4.2 Not Standardizing Apart

Recall that the FODD-based VI algorithm must add functions represented by FODDs (in Step 2) and take the maximum over functions represented by FODDs (in Step 4). Since the individual functions are independent functions of the state, the variables of different functions are not related to one another. Therefore, before adding or maximizing, the VI algorithm standardizes apart the diagrams. That is, all variables in the diagrams are given new names so they do not constrain each other. On the other hand, since the different diagrams are structurally related this often introduces redundancies (in the form of renamed copies of the same atoms) that must be be removed by reduction operators. However, our reduction operators are not ideal and avoiding this step can lead to significant speedup in the system. Here we observe that for maximization (in Step 4) standardizing apart is not needed and therefore can be avoided.

Theorem 7 *Let B_1 and B_2 be FODDs. Let B be the result of combining B_1 and B_2 under the max operation when B_1 and B_2 are standardized apart. Let B' be the result of combining B_1 and B_2 under the max operation when B_1 and B_2 are not standardized apart. $\forall$ interpretations I, MAP$_B(I) =$ MAP$_{B'}(I)$.*

Proof: The theorem is proved by showing that for any I a valuation for the maximizing diagram can be completed into a valuation over the combined diagram giving the same value. Clearly MAP$_B(I) \geq$ MAP$_{B'}(I)$ since every substitution and path that exist for B' are also possible for B. We show that the other direction holds as well. Let $\vec{u}$ be the variables common to B_1 and B_2. Let $\vec{u_1}$ be the variables in B_1 that are not in B_2 and $\vec{u_2}$ be the variables in B_2 not in B_1. By definition, for any interpretation I,

$$\text{MAP}_B(I) = Max[\text{MAP}_{B_1}(I), \text{MAP}_{B_2}(I)] = Max[\text{MAP}_{B_1}(I, \zeta_1), \text{MAP}_{B_2}(I, \zeta_2)]$$

for some valuations ζ_1 over $\vec{u}\vec{u_1}$ and ζ_2 over $\vec{u}\vec{u_2}$. Without loss of generality let us assume that MAP$_{B_1}(I, \zeta_1) = Max[MAP_{B_1}(I, \zeta_1)$, MAP$_{B_2}(I, \zeta_2)]$. We can construct valuation ζ over $\vec{u}\vec{u_1}\vec{u_2}$ such that ζ and ζ_1 share the values of variables in $\vec{u}$ and $\vec{u_1}$. Obviously MAP$_{B_1}(I, \zeta) =$ MAP$_{B_1}(I, \zeta_1)$. Also, by the definition of FODD combination, we have MAP$_{B'}(I) \geq$ MAP$_{B_1}(I, \zeta) =$ MAP$_B(I)$. ■

4.5 Discussion

This chapter introduced new reduction operators, each focusing on a particular kind of redundancy. R9 applies to equality conditions, R10 performs a global analysis based on path reachability and R11 focuses on sub-FODDs with redundant ancestors. The sub-apart subroutine improves applicability of R7. However, even with the new reduction operators, our set of reductions is heuristic and does not guarantee a canonical form for diagrams which is instrumental for efficiency of propositional algorithms. Identifying such “complete” sets of reductions operators and canonical forms is an interesting challenge. However, in spite of the elusive canonical form, as we will see in the next chapter, these new operators prove valuable in making FODD based algorithms practical.

Chapter 5

Stochastic Planning with FODDs

5.1 Introduction

The FODD based VI algorithm in Chapter 3 has been proved correct but we did not address the question of efficiency. Writing an efficient RMDP solver based on this algorithm is crucial to its practical utility. Although the issue of efficiency is partly a matter of good software engineering, there is also scope for algorithmic solutions. In the last chapter we introduced additional reduction operators that keep the FODDs compact, thereby improving efficiency. In this chapter we introduce more solutions that make the VI algorithm practical. Incorporating these, we developed FODD-PLANNER, a planning system for solving relational stochastic planning problems. The FODD-PLANNER system is evaluated on several domains, including problems from the recent international planning competition (IPC), and shows competitive performance with top ranking systems. To our knowledge this is the first application of a pure relational VI algorithm without linear function approximation to problems of this scale. As we will see later, the results demonstrate that abstraction through compact representation is a promising approach to stochastic planning.

This chapter is organized as follows. Section 5.2 describes the FODD-PLANNER system which includes several extensions of the basic algorithm and techniques for approximation. Section 5.3 presents the results of experiments on planning domains from the IPC.

5.2 FODD-PLANNER

In this section we discuss the system FODD-PLANNER that implements the VI algorithm with FODDs. FODD-PLANNER employs a number of approximation techniques that yield further speedup over the algorithms of previous chapters. The system also implements extensions of the basic VI algorithm that allow it to handle action costs and universal goals. The following sections describe these details.

5.2.1 Value Approximation

Reductions help keep the diagrams small in size by removing redundancies but when the true n step-to-go value function itself is large, legal reductions cannot help. There are domains where the true value function is unbounded. For example in the tireworld domain from the international planning competition, where the goal is always to get the vehicle to a destination city, one can have a chain of cities linked to one another up to the destination. This chain can be of any length. Therefore when the value function is represented using state abstraction, it must be unbounded. Kersting et al. (2004) and Sanner (2008) observe that SDP-like algorithms are less effective on domains where the dynamics lead to such transitive structure and every iteration of value iteration increases the size of the n step-to-go value function. In other cases the value function is not infinite but is simply too large to manipulate efficiently. When this happens we can resort to approximation keeping as much of the structure of the value function as possible while maintaining efficiency. One must be careful about the trade off here. Without approximation the run time can be prohibitive and too much approximation causes loss of structure and value. We next present three methods to get approximations that act at different levels in the algorithm.

Not Standardizing Apart Action Variants

As shown in Section 3.8.2, standardizing apart of action variant diagrams before adding them is required for the correctness of the FODD based VI algorithm. That is, if we do not standardize apart action variant diagrams before adding them, the value given to some states may be lower than the true value. Intuitively, this is true since different paths in the value function share atoms and variables. Now, for a fixed action, the best variable binding and corresponding value for different action variants may be different. Thus, if the variables are forced to be the same for the variants, we may rule out viable combinations of value. On the other hand, *the value obtained if we do not standardize apart is a lower bound on the true value*. This is because every path in the diagram resulting from not standardizing apart is present in the diagram resulting from standardizing apart. On the other hand, not standardizing apart leads to more compact diagrams. We call this approximation method *non-std-apart* and use it as a heuristic to speed up computation. Although this heuristic may cause loss of structure in the representation of the value function, we have observed that in practice it gives significant speedup while maintaining most of the relevant structure.

Merging Leaves

The use of FODDs also allows us to approximate the value function in a simple and controlled way. Here we follow the approximation techniques of APRICODD (St-Aubin et al., 2000) where they were used for propositional problems. The idea is to reduce the size of the diagram by merging substructures that have similar values. One way of doing this is to reduce the precision of the leaf values. That is, for a given (user defined) precision value ϵ, we join leaves whose value is within ϵ. This can cause reduction of the diagram because sub-parts of the diagram that previously pointed to different leaves, now point to the same leaf. The granularity of approximation, however, becomes an extra parameter for the system and has to be chosen carefully.

Domain Determinization

Previous work on stochastic planning has discovered that for some domains one can get good performance by pretending that the domain is deterministic and replanning if unexpected outcomes are reached (Yoon et al., 2007). For this approximation, we use a similar idea and determinize the domain in the process of policy generation. This saves significant amount of computation and avoids the typical increase in size of the value function encountered in step 2 of the VI algorithm. Domains can be determinized in many ways.

In our experiments we chose to perform determinization by replacing every stochastic action with its most probable deterministic alternative. Although this is not always the ideal method of determinization, it makes sense when the most probable outcome corresponds to the successful execution of an action (Little & Thibaux, 2007) as in the case of the domains we experimented on. Determinization is done only once prior to running VI. Note that the determinization only applies to the process of policy generation. When the generated policy is deployed to solve planning problems, it does so under the original stochastic environment.

5.2.2 Extensions to the VI Algorithm

FODD-PLANNER makes two additional extensions to the basic algorithm. This allows the handling of action costs, arbitrary conjunctive goals as well as universal goals.

Handling Action Costs

The standard way to handle action costs is to replace $R(s, a)$ by $R(s, a) - Cost(a)$ in the VI algorithm. However, our formalism using FODDs relies on the fact that all the leaves (and thus values) are non-negative. To avoid this difficulty, we note that action costs can be supported as long as there is at least one zero cost action. To see this recall the VI algorithm. The appropriate place to add action costs is just before the Object Maximization step. However, because this step is followed by maximizing over the action diagrams, if at least one action has 0 cost (if not we can create a *no-op* action), the resultant diagram after maximization will never have negative leaves. Therefore we safely convert negative leaves before the maximization step to 0 and thereby avoid conflict with the reduction procedures.

Heuristically Handling Universal Goals

FODDs with max aggregation cannot represent universal quantifiers. Therefore our VI algorithm cannot handle universal goals at the abstract level (in Chapter 8 we develop a formalism that does accept arbitrary quantifiers). For a concrete planning problem with a known set of objects we can instantiate the universal goal to get a large conjunctive goal. In principle we can run VI and policy generation for this large conjunctive goal. However,

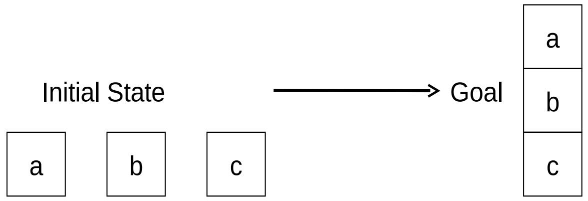


Figure 5.1: Example where the AGD heuristic awards equal value to the optimal and sub-optimal actions

this would mean that we cannot plan offline to get a generic policy and must replan for each problem instance from scratch. Here we follow an alternative heuristic approach previously introduced by Sanner and Boutilier (2006) and use an approximation of the true value function, that results from a simple additive decomposition of the goal predicates.

Concretely, this additive goal decomposition (AGD) heuristic works as follows. During offline planning we plan separately for a generic version of each predicate. Then at execution time, when given a concrete goal, we approximate the true value function by the sum of the generic versions over each ground goal predicate. This is clearly not an exact calculation and will not work in every case. On the other hand, it considerably extends the scope of the technique and works well in many situations.

However, this heuristic is limited and does not apply well to domains where goal literals must be achieved in some order. As an example consider a problem in the well known blocksworld domain. In this domain the world consists of blocks on a table in some configuration. The objective is to rearrange the blocks into some other specified configuration by picking them and putting them down. Figure 5.1 shows a simple example problem in this domain. Clearly the block b has to be put on block c before a is put on b. However, the AGD heuristic adds up the values for each individual goal resulting in the effect that the action that puts a on b first and the one that puts b on c first get equal value. This results in the wrong action being chosen half of the time. For problems with a larger domain this can cause failure of plan execution.

To address this issue, we propose a new heuristic based on weighted goal ordering (WGO). The idea is to first get a partial ordering between each pair of goal literals. We use the following heuristic. For each pair of goal literals g_1 and g_2 we check if $\neg g_2$ is a precondition for some action to bring about g_1. If this condition is satisfied, g_1 must be achieved before g_2 in the partial order. The intuition here is that preconditions expose some obvious ordering constraints on the goals. Identifying all such constraints amounts to solving a deterministic planning problem. Instead the heuristic identifies constraints that we can discover easily.

Respecting this partial order we impose a total order on the goals in an arbitrary way. Finally the value of an action is calculated by adding the values of the individual goals literals as before, except that this time we weight the values of the individual sub-goals in proportion to their position in the total order. The value of the goal literal at position i is weighted by w^{i-1}. The weight parameter w ($0 < w \leq 1$) is user defined. As expected, smaller weights are better for domains with interacting sub-goals and large weights are better for domains where sub-goals are independent. In the example in Figure 5.1, an action that puts a on b will get a lower value than the action that puts b on c in the start state. These ideas help serialize any obvious ordered set of goals and gives a weak preference ordering on other sub-goals. This leads to significant improvements in performance in domains with interacting dependent sub-goals as we show in the next section.

5.2.3 The FODD-Planner System

We implemented the FODD-PLANNER system, plan execution routines and evaluation routines under Yap Prolog 5.1.2. Our implementation uses a simple theorem prover that supports background knowledge by "state flooding". That is, to prove $\mathcal{B} \models X \rightarrow Y$, where X is a ground conjunction (represented in prolog as a list), we "flood" X using rules of the background knowledge using the following simple steps until convergence.

1. Generate Z, the set of all ground literals that can be derived from X and the rules of background knowledge.

2. Set $X = X \cup Z$.

When X has converged we test for membership of Y in X. Because of our restricted language, the reasoning problem is decidable and our theorem prover is complete. An alternative to the list representation of X would have been to utilize the prolog database to store the literals of X and employ the fast prolog engine to query Y. However, in our experience with Yap, it becomes expensive to *assert* (and *retract*) the literals of X to (from) the prolog database so that the list representation is faster.

The overall algorithm is the same as SDP except that all operations are performed on FODDs and reductions are applied to keep all intermediate diagrams compact. In the experiments reported below, we use all previously mentioned reductions (R1 $\cdots$ R11) except R7-replace. We applied reductions iteratively until no reduction was applicable on the FODD. There is no correct order to apply the reductions in the sense that any reduction when applied can give rise to other reductions. Heuristically we chose an order where we hope to get as much of the diagram reduced as soon as possible. We apply reductions in the following order. We start by applying R10 twice with a different DPO each time. The first DPO is generated by breaking ties in favor of shorter paths. The second is generated by reversing the order of equal valued paths in the first DPO. With R10 we hope to catch many redundant edges early. R10 is followed by R7-drop to remove redundant nodes connected to the edges removed by R10. After this, we apply a round of all strong reductions followed by R9 to get rid of the redundant equality nodes. R9 is followed by another round of strong reductions. This sequence is performed iteratively until the diagram is stable. In effect strong reductions are applied every time two diagrams are combined and weak reductions are applied every time two diagrams are combined except during regression by block combination. We chose to apply R11 only twice in every iteration - once after regression and once just before the next iteration. This setting for application of reduction operators is investigated experimentally and discussed in Section 5.3.1.

5.3 Experimental Results

We ran experiments on the logistics problem as well as probabilistic planning domains from the international planning competitions (IPC) held in 2006 and 2008. All experiments were run on a Linux machine with an Intel Pentium D processor running at 3 GHz, with 2 GB of memory. All timings, rewards and plan lengths we report are averages over 30 rounds. For each domain, we constructed by hand background knowledge restricting arguments of predicates (e.g., a box can only be at one city in any time so $Bin(b, c_1), Bin(b, c_2) \rightarrow (c_1 = c_2)$). This is useful in the process of simplifying diagrams. The domain definition (TVDs

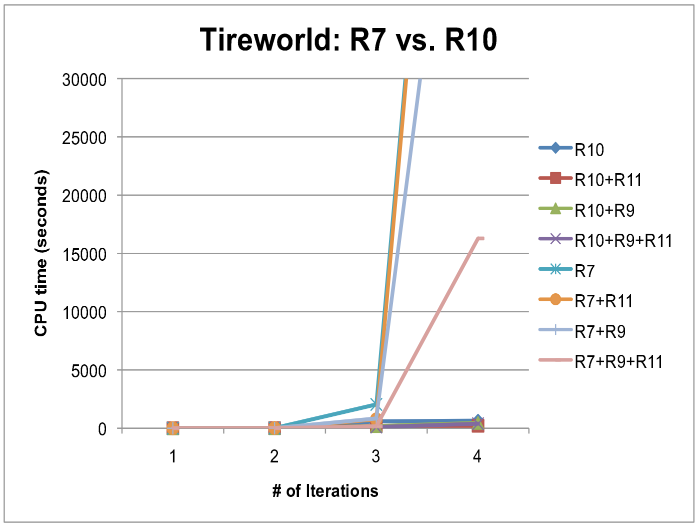


Figure 5.2: A comparison of planning time taken by various settings of reduction operators over varying number of iterations. Four settings of R10 are compared against four settings of R7.

and reward function diagrams) was generated by translating the IPC encoding, which is given in the PPDDL language (Younes et al., 2005), manually. To avoid long execution times, we set a limit of 200 steps on the plan length when solving IPC problems. Plans that ran for more than 200 steps were counted as failed.

5.3.1 Merits of Reduction Operators

In our first set of experiments we used the tireworld domain to compare the relative merits of the new reductions R9, R10 and R11 along with R7. The experimental setup was the same as detailed in Section 5.2.3. Since R10 and R7 are both edge removal reductions and R7-drop is used in conjuction with both, we compare R10 to R7-replace directly under all configurations of R9 and R11. Figure 5.2 shows the time to build a policy over varying number of iterations for different settings of these weak reduction operators. Strong reductions were always applied. The figure clearly shows the superiority of R10 over R7-replace. All combinations with R7-replace have prohibitively large run times at 3 or 4 iterations. With or without R9 and R11, R10 is orders of magnitude more efficient than R7-replace.

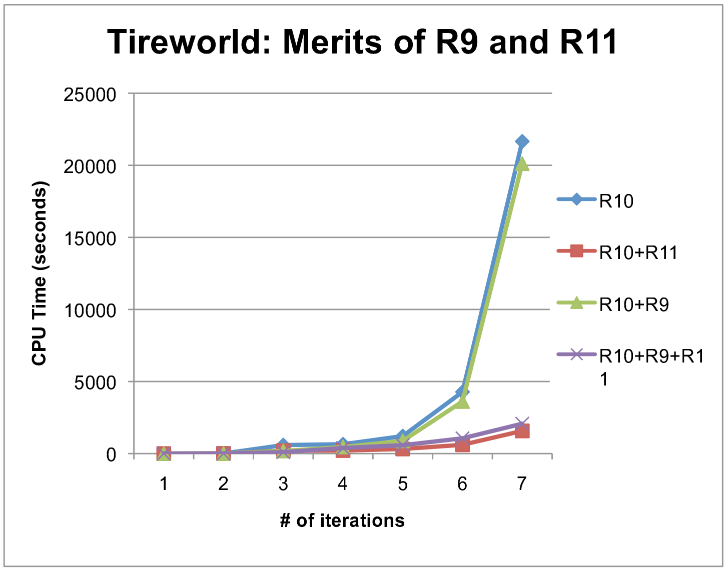


Figure 5.3: A comparison of the merits of R9 and R11 in the presence of R10

It is for this reason that in all future experiments we used R10 instead of R7-replace. Figure 5.3 shows the relative merits of R9 and R11 in the presence of R10. Clearly R11 is an important reduction and it makes planning more efficient in both settings (just R10 and R10+R9). Another fact, while not evident from Figure 5.3, is that R9 improves planning performance when the number of iterations is small (1 to 3). This makes sense because R9 eliminates equality nodes and combines the sub-FODDs under them. When diagrams get larger (as with more iterations), combinations can produce even larger diagrams and slow down reductions. Even so, R9 can help in not infrequent cases where the combination does not produce a larger diagram. At the same time, given a particular setting (just R10 or R10+R11) the addition of R9 does not cause a significant drop in performance even for more iterations. In addition, R9 targets the removal of equality nodes which no other reduction does directly. Based on these results we choose the setting where we employ R10 along with R9 and R11 for the experiments henceforth.

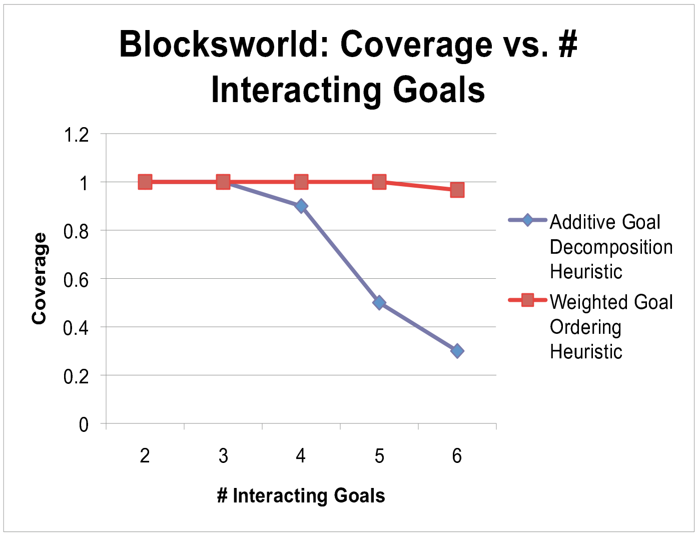


Figure 5.4: WGO Heuristic vs. AGD Heuristic

5.3.2 The Logistics Benchmark Problem

This is the running example introduced in Chapter 1. Because of the assumption that all cities are reachable from each other this domain has a compact abstract optimal value function. Like ReBel (Kersting et al., 2004) and FOADD (Sanner, 2008) we were able to solve this RMDP and identify all relevant partitions of the optimal value function and in fact the value function converges after 10 iterations. FODD-PLANNER performed 10 iterations in under 2 minutes.

5.3.3 Conjunctive Goals and Goal Ordering

Before presenting results on IPC benchmarks we demonstrate the difference between the AGD and the WGO heuristic. The difference between the two becomes apparent in domains where the order in which goal literals are achieved is important. To test this we generated problems with 5 blocks in the blocksworld domain with increasing number of interacting goals. This was done by manually constructing goal states with towers of increasing height. Figure 5.4 plots the percentage of planning problems solved against the number of interacting goal literals on these problems where we set the parameter w to 0.8. We observe that as the number of interacting goals increases, the AGD heuristic is

	Coverage	Time (ms)	Reward
GPT	100%	2220	57.66
Policy Iteration with policy language bias	46.66%	60466	36
Re-Engg NMRDPP	10%	290830	-387.7
FODD-Planner	100%	65000	47.3

Table 5.1: Fileworld domain results

out-performed by the WGO heuristic. In domains where the order of achievement of goal literals is not crucial to solving the planning problem, the WGO heuristic has no effect. For such domains we set the w parameter to 1.0 causing the WGO heuristic to fall back on the AGD heuristic.

5.3.4 The Fileworld Domain

This domain was part of the probabilistic track of IPC-4 (2004) (http://ls5-web.cs.uni-dortmund.de/∼edelkamp/ipc-4/). The domain consists of files and folders. Every file obtains a random assignment to a folder at execution time and the goal is to place each file in its assigned folder. There is a cost of 100 to handle a folder and a cost of 1 to place a file in a folder. Results have been published for one problem instance which consisted of thirty files and five folders. The optimal policy for this domain is to first get the assignments of files to folders and then handle each folder once, placing all files that were assigned to it. Because the goal is conjunctive we used the additive goal decomposition discussed above. We used offline planning for a generic goal $filed(a)$ and use the policy to solve for any number of files. This domain is ideal for abstract solvers since the optimal value function and policy are compact and can be found quickly. The FODD-PLANNER was able to achieve convergence within 4 iterations even without approximation. Policy generation and execution together took 65 seconds. Of the 6 systems that competed on this track, results have been published for 3 on the website cited above. Table 5.1 compares the performance of FODD-PLANNER to the others. We observe that we rank ahead of all except GPT in terms of total reward and coverage (both FODD-PLANNER and GPT achieve full coverage). We do not get perfect reward in spite of a converged exact value function because of the additive decomposition of rewards. The policy generated is optimal for one

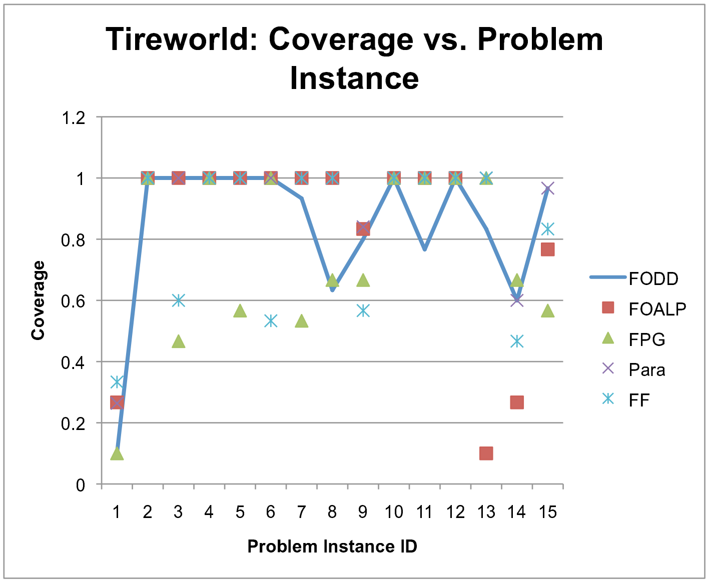


Figure 5.5: Coverage result of tireworld experiments

file but it is still a heuristic for many files. The order of goal literals is not important and so the w parameter was set to 1.0 in these experiments.

5.3.5 The Tireworld Domain

This domain was part of the probabilistic track of IPC-5 (2006) The domain consists of a network of locations (or cities). A vehicle starts from one city and moves from city to city with the objective of reaching a destination city. Moves can only be made between cities that are directly connected by a road. However, on any move the vehicle may lose a tire with 40% probability. Some cities have a spare tire that can be loaded onto the vehicle. If the vehicle contains a spare tire, the flat tire can be changed with 50% success probability. This domain is simple but not trivial owing to the possibility of a complex network topology and high probabilities of failure. Participants at IPC-5 competed over 15 problem instances on this domain. To limit offline planning time we restricted FODD-PLANNER to 7 iterations without any approximation for the first 3 iterations and with the non-std-apart approximation for the remaining iterations. The policy was generated in 2.5 hours. The goal always contains a single literal describing where the vehicle should be. Therefore

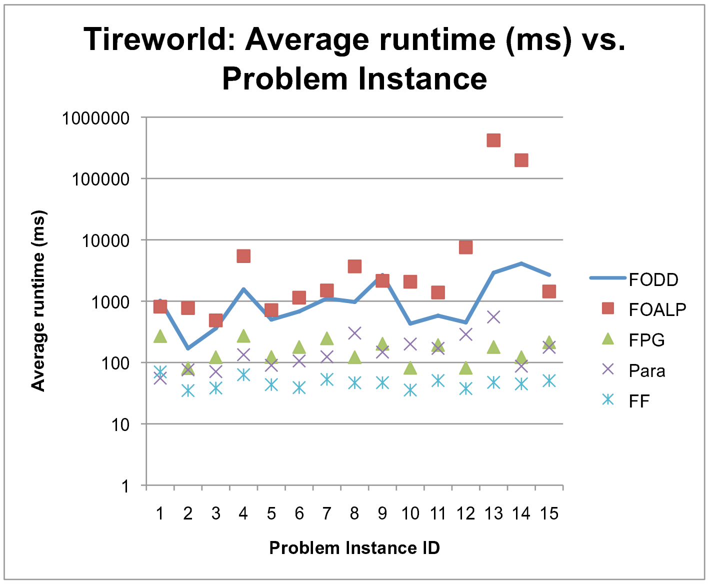


Figure 5.6: Timing result of tireworld experiments

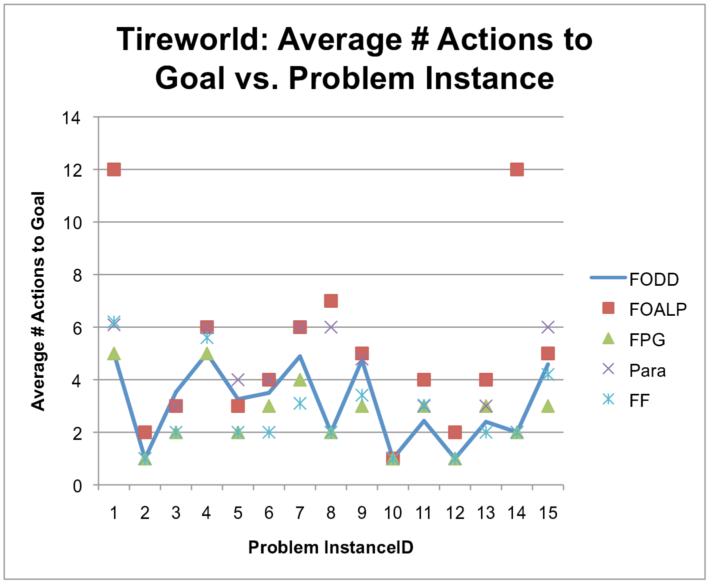


Figure 5.7: Plan length result of tireworld experiments

Precision	Planning Time	Execution Time (sec)	Coverage	Plan length
50	81.89%	87.77	13.99	7.47
75	95.10%	90.33	15.48	10.40
100	95.13%	90.52	15.48	10.40
125	99.37%	94.78	32.99	-30.93
150	99.47%	94.88	32.99	-30.93

Table 5.2: Percentage average reduction in planning time, execution time, coverage and plan length, for tireworld under the merging of leaves approximation for varying leaf precision values. For example, the first row of the table states that by reducing the precision on the leaves to 50, which is 10% of the largest achievable reward in any state, the planning time was reduced by 81.89% of its original value, average execution time was reduced by 87.77%, average coverage was reduced by 13.99% and average plan length was reduced by 7.74%

ordering goal literals is not relevant. The performance of FODD-PLANNER and systems competing in the probabilistic track of IPC-5, for which, data is published, is summarized in Figures 5.5, 5.6, and 5.7. The figures show a comparison of the percentage of problem instances each planner was able to solve (coverage), the average time per instance taken by each planner to generate an online solution, and the average number of actions taken by each planner to reach the goal on every instance. We observe that the overall performance of FODD-PLANNER is competitive with (and in a few cases better than) the other systems. Run Times to generate online solutions are high for FODD-PLANNER but are comparable to FOALP which is the only other First-Order planner. Overall run time of our system (offline plus online) is within the time limit of the competition. On the other hand, in comparison with the other systems, we are able to achieve high coverage and short plans on most of the problems.

Although this domain can be solved as above within the IPC time, one might wish for even faster execution. As we show next, the heuristic of merging leaves provides such a tool, potentially trading off quality of coverage and plan length for faster planning and execution times. Table 5.2 shows the average reduction in planning time, coverage and planning length achieved when the approximation of merging leaves is used. The highest reward obtained in any state is 500. We experimented with reducing precision on the leaves from 50.0 and 150.0. As the results demonstrate, for some loss in coverage and planning length, the system can gain in terms of execution time and planning time. For example,

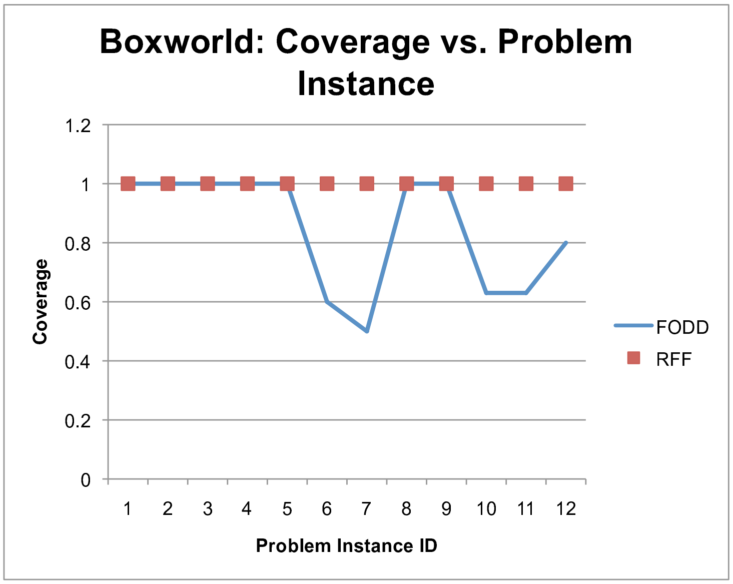


Figure 5.8: Coverage results of boxworld experiments

with leaf precision of 10% we get 81.89% reduction in planning time (5 fold speedup) but we lose 13.99% in coverage.

5.3.6 Boxworld

In this domain from IPC 2008, the world consists of boxes, trucks, planes and a map of cities. The objective is to get boxes from source cities to destination cities using the trucks and planes. Boxes can be loaded and unloaded from the trucks and planes. Trucks (and planes) can be driven (flown) from one city to another as long as there is a direct road (or air route) from the source to the destination city. The only probabilistic action is *drive*. *drive* works as expected (transporting the truck from the source city to the destination city) with probability 0.8. With probability 0.2 *drive* teleports a truck to the wrong city.

IPC posted 15 problems with varying levels of difficulty for this domain. Competition results show that RFF (Teichteil-Koenigsbuch et al., 2008) was the *only* system that solved any of the 15 problems. Neither RFF nor FODD-Planner could solve problems 13 to 15. Hence we omit results for those.

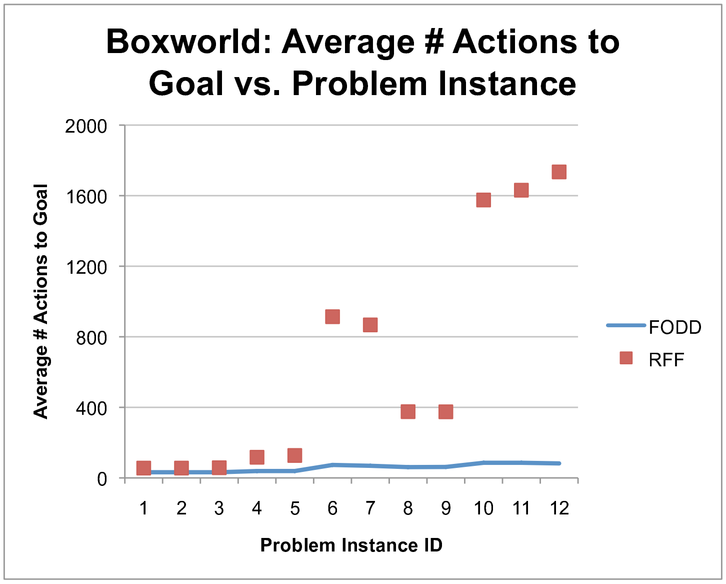


Figure 5.9: Plan length results of boxworld experiments

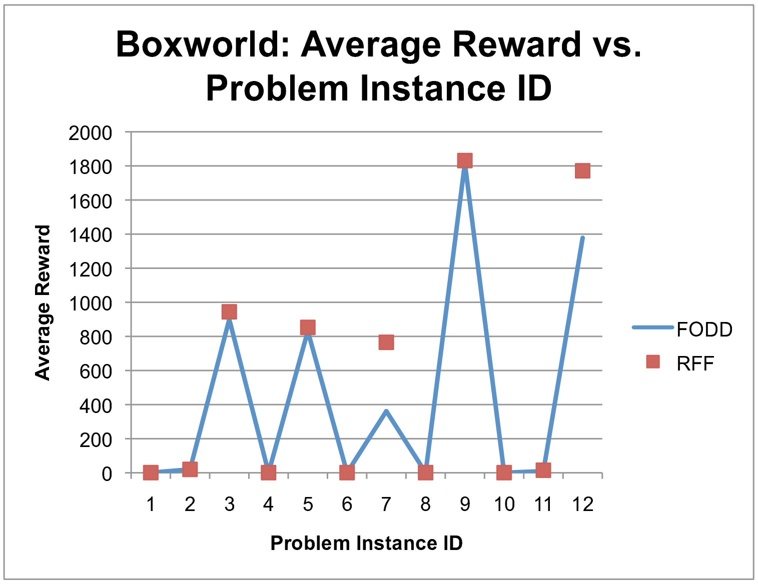


Figure 5.10: Average reward results of boxworld experiments

To limit offline planning time we determinized this domain (making *drive* deterministic) and restricted FODD-PLANNER to 5 iterations. Since the domain was determinized, there was only one alternative per action. Therefore the the non-std-apart approximation has no effect here. The policy was generated in 55.5 minutes. Goal literals in this domain can be achieved independently of each other. Therefore goal ordering is not effective and the w parameter was set to 1.0. The performance of FODD-PLANNER and RFF is summarized in Figures 5.8, 5.9, and 5.10. The figures show a comparison of the percentage of problem instances each planner was able to solve (coverage), the average reward achieved per problem instance, and the average number of actions taken by each planner to reach the goal on every instance.

As can be seen FODD-PLANNER has lower coverage than RFF on problems 10, 11 and 12. However, our performance is close to RFF in terms of accumulated reward and consistently better in terms of plan length even on problems where we achieve full coverage[1]. In this domain we experienced extremely long plan execution times (1.5 hours per round on hard problems and about 100 seconds per round on the easier problems). This could be a one reason for the failure of other planning systems at IPC where a strict time bound was observed, and for the failure of RFF on problems 13, 14 and 15. Improving run time of the online application of our policies is an important aspect for future work. One direction might be to employ a more efficient subsumption (matching a rule to the state) routine. Another possibility is to cache queries to the policy FODD.

In this domain the heuristic of merging leaves did not provide any advantage. As in tireworld, there is a clear trade off between the quality of coverage and planning time. However the loss in coverage is very high for the planning efficiency gained. Additionally, this heuristic does not reduce execution time, which is the main bottleneck in this domain.

5.3.7 Blocksworld

This is the classic domain discussed in the context of goal ordering above. This domain has many variants. In IPC 2006 this domain was described by 7 probabilistic actions. For the

[1] When coverage is not full it is possible that a system solving only easy problems can look better in terms of planning length (because their solutions are shorter) and therefore average planning length is not a good criterion for comparison. However with full coverage planning length provides a valid comparison

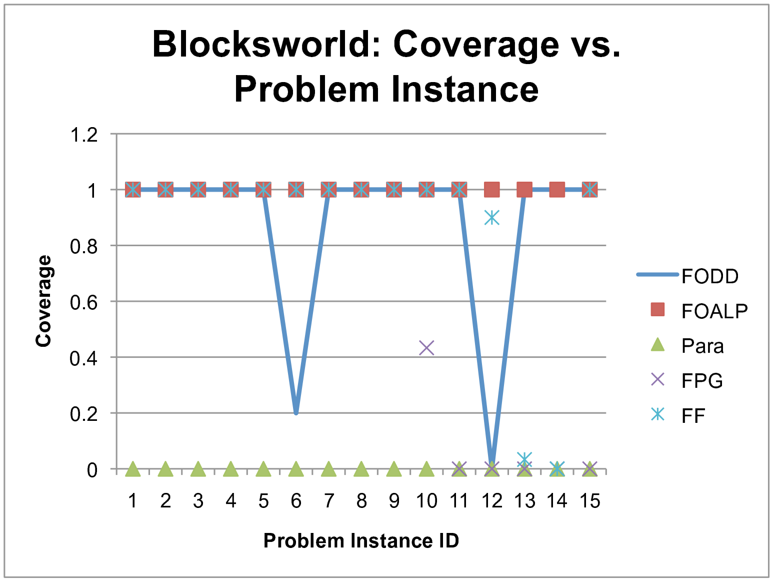


Figure 5.11: Blocksworld Coverage Results

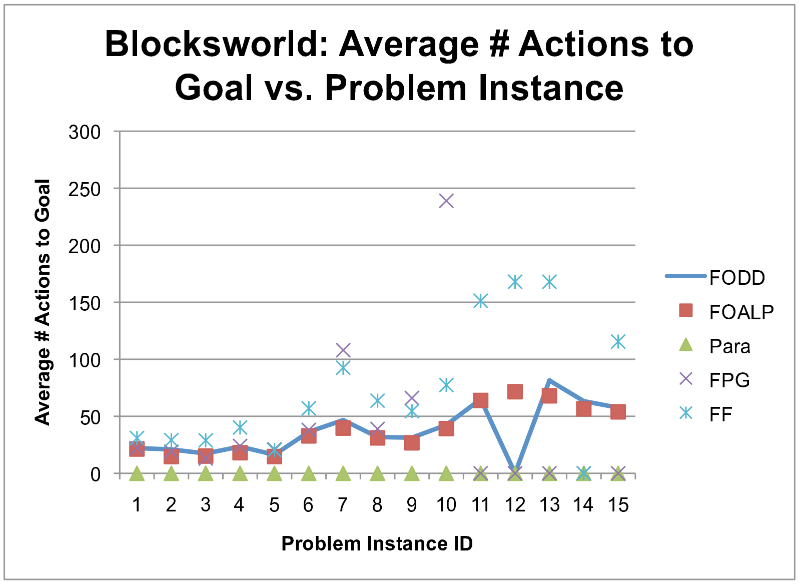


Figure 5.12: Blocksworld Plan Length Results

purpose of VI we simplified the domain by determinizing it. The resultant blocksworld domain consisted of 4 deterministic actions - pick-up-block-from-table, pick-up-block-from-block, put-block-on-table, put-block-on-block.

We limited FODD-PLANNER to 8 iterations to stay within competition times. The policy was generated in 3.46 hours. IPC posted 15 problems with varying degrees of difficulty. The goal is each of these problems is a configuration of blocks consisting of a set of towers. The goal state, therefore, consists of multiple interacting goal literals. Hence, the goal ordering heuristic is useful in this domain. In our experiments we achieved no coverage with the AGD heuristic but full coverage on almost all of the problems by switching to the WGO heuristic. The w parameter was heuristically set to 0.8. Figures 5.11 and 5.12 show the comparison of cover and plan length of the FODD-PLANNER with systems competing in IPC 2006. FODD-PLANNER achieves performance close to the top rankers in terms of both metrics. We experienced long plan execution times, similar to the boxworld problems, for the harder problems (10 to 15) in this domain. Again, improving this aspect of the system is left to future work. The heuristic of merging leaves was not helpful in this domain.

5.4 Summary and Concluding Remarks

The main contribution of this chapter is the introduction of FODD-PLANNER, a relational planning system based on FODDs as the underlying representation. This is the first planning system that uses lifted algebraic decision diagrams as its representation language and successfully solves planning problems from the IPC.

We also introduced the WGO heuristic for goal ordering in RMDPs. Although WGO requires setting the weight parameter, w, it performs better than previous heuristics on domains where goal literals have to be achieved in a certain order. In exploratory experiments we observed that the performance of FODD-PLANNER is consistent over large parts of the weight space. Therefore future work may include experiments where the w parameter is learned or set by cross-validation.

The experimental results show that abstraction through compact representation is a promising approach to stochastic planning. They also raise many interesting questions concerning foundations for FODDs and their application to solve RMDPs. One important aspect is the question of reductions. Our set of reductions is still heuristic and does not guarantee a canonical form for diagrams which is instrumental for efficiency of propositional algorithms. Identifying such “complete” sets of reductions operators and canonical forms is an interesting challenge. Identifying a practically good set of operators trading off complexity for reduction power is crucial for further applicability. In the next chapter we develop novel methods for improving the applicability and efficiency of reductions to achieve even better performance.

Chapter 6

Model-Checking Reductions

One characteristic of systems based on the SDP algorithm and in general systems that reason with relational representations is the need for logical simplification of formulas. In SDP based systems, like our FODD-PLANNER for example, backward reasoning introduces redundancies in the structure of the value function, creating a need for logical simplification in order to maintain a value function of reasonable size. The simplification steps are at the core of SDP based systems and they must be implemented efficiently. To date, all such systems (Kersting et al., 2004; Hölldobler et al., 2006; Sanner & Boutilier, 2009) have employed theorem proving to identify and remove redundancies. This is also true for the FODD-PLANNER as described in the previous chapter. However, theorem proving comes with two major drawbacks.

- Firstly, it is expensive.

- Secondly, in all but trivial domains, reduction effectively requires the theorem prover to have access to some background knowledge from the domain. For example in the logistics domain, knowledge that a box cannot simultaneously be in more than one city has to be supplied to the theorem prover. By encoding detailed background knowledge, more redundancies can be identified but this increases the run time of the theorem proving routine. On the other hand, if one encodes too little background knowledge, it is impossible to identify redundant structures in the value function and when the value function is large, reasoning becomes slow. This trade-off has to be balanced by encoding just the right amount of background knowledge, a challenging task when we aim for domain independent solutions.

In this chapter we show that both of these issues can be mitigated by changing the focus to model-checking reductions. More precisely, we present the idea of model-checking reductions and give an algorithm which is sound and complete for reducing FODDs. Next we design practical variants of this algorithm thus developing approximate but efficient reductions for FODDs.

6.1 R12: The Model Checking Reduction for FODDs

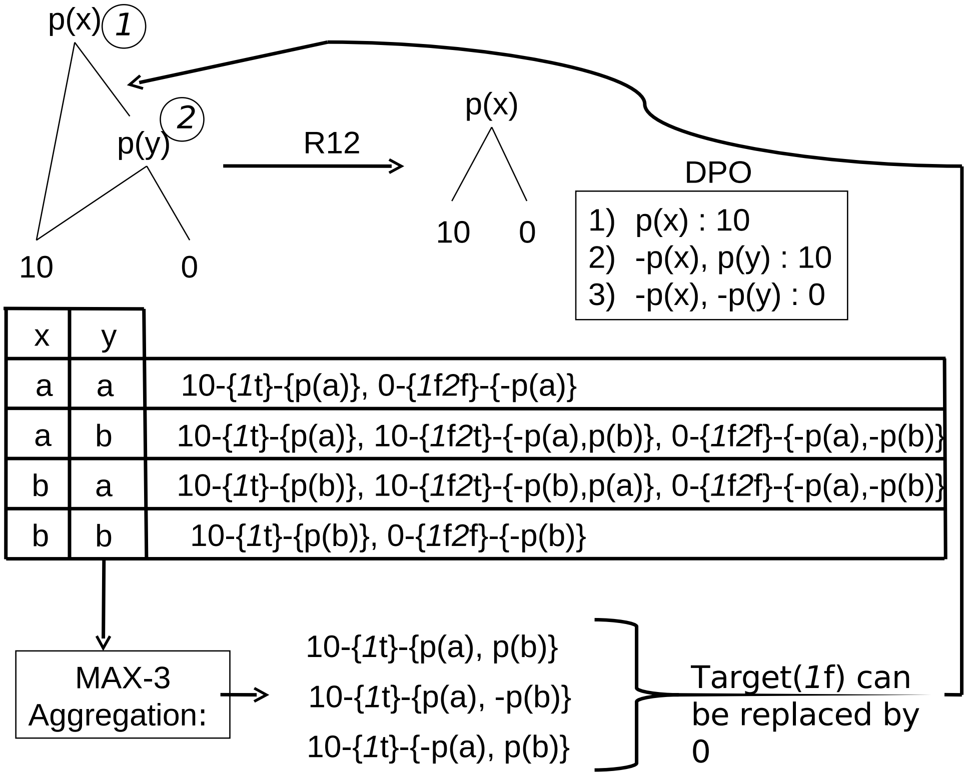
In this section we introduce a new reduction operator R12. The basic intuition behind R12 is to use the semantics of the FODD directly in the reduction process. According to the semantics of FODDs the map is generated by aggregation of values obtained by running all possible valuations through the FODD. Therefore, if we run all possible valuations through the diagram and document the behavior of valuations under all possible interpretations, we can identify parts of the diagram that are never important for determining the map. Such parts can then be eliminated to reduce the diagram. Crucially, with some bookkeeping, it is possible to obtain this information without enumerating all possible interpretations and by enumerating all possible valuations over just the variables in the diagram.

Enumeration of all possible interpretations can be avoided with the observation that although there can be many interpretations over a set of domain objects, there are only a fixed number of paths in the FODD that a valuation can traverse. For a given valuation ζ, any interpretation can be classified into one of a set of equivalence classes based on the path p that it forces ζ through. All interpretations belonging to an equivalence class have the following in common.

1. They force ζ through path p.

2. They force ζ to the same leaf - leaf(p).

3. They are consistent with PF(p)(ζ).



x	y	
a	a	10-{1t}-{p(a)}, 0-{1f2f}-{-p(a)}
a	b	10-{1t}-{p(a)}, 10-{1f2t}-{-p(a),p(b)}, 0-{1f2f}-{-p(a),-p(b)}
b	a	10-{1t}-{p(b)}, 10-{1f2t}-{-p(b),p(a)}, 0-{1f2f}-{-p(a),-p(b)}
b	b	10-{1t}-{p(b)}, 0-{1f2f}-{-p(b)}

Figure 6.1: An example of reduction operator R12 for FODDs. Each entry of the form value-{path}-{interpretation} in the table expresses the value obtained by running the valuation of the corresponding row through the diagram under an equivalence class of interpretations. The MAX-3 aggregation function then calculates the possible aggregates that could be generated under different equivalence classes of interpretations. Since the edge *1f* does not appear along any of the paths leading to a non-zero leaf in the result of MAX-3, it is not crucial towards determining the map and can be removed

PF(p)(ζ) is, thus, the most general interpretation that forces ζ through p and can be viewed as a key or identifier for its equivalence class. For the purpose of reduction we are not interested in the interpretations themselves but only in the paths that they force valuations through. Therefore we can restrict our attention to the equivalence classes and avoid enumerating all possible interpretations. In other words, if we collect the abstract interpretation PF(p)(ζ) for every path p that a valuation ζ could possibly take (i.e. every path where PF(p)(ζ) is consistent), along with the corresponding path and leaf reached, we will have all information we need to describe the behavior of ζ under all possible interpretations. The procedure *getValue* described below, does exactly that by simulating the run of a valuation through a FODD. The output of the procedure is a set of $\langle leaf$, EL, $I\rangle$ triplets, where $leaf$ is the leaf reached by the valuation ζ by traversing the path p (described by the set of edges EL) and $I =$ PF(p)(ζ). We run *getValue* for valuation ζ with root node n and empty sets for PF and EL

Procedure 5 *getValue(valuation ζ, PathFormula PF, EdgeList EL, Node n)*

```
1. If $n$ is a leaf, return $\{\{l(n), EL, PF\}\}$

2. If $\mathcal{B} \models PF \rightarrow l(n)(\zeta)$, then return getValue($\zeta$, $PF \cup l(n)(\zeta)$, $EL \cup n_{\downarrow t}$, target($n_{\downarrow t}$))
   If $\mathcal{B} \models PF \rightarrow \neg l(n)(\zeta)$, then return getValue($\zeta$, $PF \cup \neg l(n)(\zeta)$, $EL \cup n_{\downarrow f}$, target($n_{\downarrow f}$))
   Return getValue($\zeta$, $PF \cup l(n)$, $EL \cup n_{\downarrow t}$, target($n_{\downarrow t}$)) $\cup$ getValue($\zeta$, $PF \cup \neg l(n)$, $EL \cup n_{\downarrow f}$, target($n_{\downarrow f}$))
```

Figure 6.1 shows an example of the R12 reduction. The reduction is applied to the FODD on the left to reduce it to the FODD on the right. The table illustrates the result of running the getValue procedure on all possible valuations over the set of domain objects $\{a, b\}$ and the variables x and y appearing in the left FODD. For example, the traversal of valuation $\{x/a, y/b\}$ through the FODD has 3 possible eventualities. Either it reaches a 10 leaf by traversing path $\{1t\}$ (which is short for *path consisting of the true edge of node* 1), under abstract interpretation $\{p(a)\}$, or it reaches a 10 leaf by traversing path $\{1f2t\}$ (which is short for *path consisting of the false edge of node* 1 *followed by the true edge of node 2*), under abstract interpretation $\{\neg p(a), p(b)\}$ or (in all other cases) it reaches a 0 leaf.

Note that the different behaviors of a valuation are mutually exclusive because the abstract interpretations associated with these behaviors partition the space of worlds. Any interpretation must be consistent with exactly one of these abstract interpretations and hence must force the behavior corresponding to that abstract interpretation on the valuation.

Thus with the help of the getValue procedure, the possible behaviors of all valuations over a set of domain objects can be tabulated. The next step is to generate all possible ways in which an aggregate value can be derived. This can be done without enumerating all interpretations. The table gives sufficient information to list all possible ways to aggregate over the set of all valuations. Just consider all combinations of behaviors over the set of valuations. Every combination (as long as it is consistent) can produce an aggregate value or the map.

The aggregation, however, has to be done so as to expose the valuations (and thereby the paths) that prove to be important for determining the map. Intuitively, then, paths that remain unexposed in spite of listing all possible ways to aggregate over the set of all valuations are unimportant and can be removed. To this end, the next section introduces variants of the max aggregation function denoted max^2 and max^3.

6.1.1 Generalized Aggregation Function and the R12 Reduction

When calculating the map, the max aggregation operation is applied to values obtained by evaluating the FODD under different valuations. For R12, we are interested not just in the aggregate value but also in other information related to the aggregate value. This information could be

1. The valuations that were indispensable in generating the aggregate value.

2. The paths followed by those valuations.

3. The minimal interpretation under which the valuations were important.

In max aggregation, only one valuation is indispensable in generating the aggregate $-$ the valuation corresponding to the highest value (if it is unique). If all values below the highest were to be removed from the input set max aggregation would still return the same result. Therefore none of the removed values are important for determining the result. Recall from Chapter 4 that a path is instrumental relative to a given DPO if

1. there is an interpretation I and valuation, ζ, such that $Path_B(I, \zeta) = p_j$, and

2. $\forall$ valuations η, if $Path_B(I, \eta) = p_k$, then $k \geq j$.

Thus valuations that are indispensable towards generating an aggregate value follow instrumental paths. Removal of such paths can cause the map to change. All non-instrumental paths, however, can be safely removed.

Algorithmically we define three variants of the max aggregation operator.

max^1**:** The first variant max^1 is the usual aggregation operator that given a set of values $\{v_1, \cdots v_n\}$ returns the aggregate $v = max(\{v_1, \cdots v_n\})$.

max^2**:** The second variant max^2 is defined relative to a DPO as follows. Let PL be a fixed DPO. Under PL, $max^2(\{\langle v_1, path_1, I_1\rangle, \langle v_2, path_2, I_2\rangle, \cdots \langle v_n, path_n, I_n\rangle\}) = \langle v_o,$ $path_o,$ $I_o\rangle$. The input to the second variant is, therefore, a set of 3-tuples of the form $\langle v_i, path_i, I_i\rangle$. Each 3-tuple corresponds to a valuation ζ_i so that ζ_i traverses path $path_i$ in the FODD under interpretation I_i (and therefore all consistent extensions of interpretation I_i) to reach leaf v_i. The output is a 3-tuple defined as:

1. $v_o = max^1(\{v_1, v_2 \cdots v_n\})$.

2. $I_o = \bigcup_{i=1}^{n} I_i$.

3. $path_o = path_i$ such that $v_i = v_o$ and $i = min\{j \mid v_j = v_0\}$.

The example in Figure 6.1 shows the DPO and the 3 possible aggregation results derived from the table. Each of the 3 results is derived using the max^2 variant. For example, aggregating over

- $\langle 10, \{1t\}, \{p(a)\}\rangle$ for $\{x/a, y/a\}$,

- $\langle 10, \{1t\}, \{p(a)\}\rangle$ for $\{x/a, y/b\}$,

- $\langle 10, \{1t\}, \{p(b)\}\rangle$ for $\{x/b, y/a\}$, and
- $\langle 10, \{1t\}, \{p(b)\}\rangle$ for $\{x/b, y/b\}$,

using the max^2 variant gives $\langle 10, \{1t\}, \{p(a), p(b)\}\rangle$ indicating that there is a possible aggregation where the path consisting of the edge $\{1t\}$ is instrumental in determining the map. Note that the resulting partial interpretation $\{p(a), p(b)\}$ is consistent but this does not have to be the case.

max^3**:** The third variant max^3 gets as input a set T. Each element of T is a $\langle valuation - valueset\rangle$ pair. A $valueset$ is itself a set of $\langle value, path, Interpretation\rangle$ triplets. In Figure 6.1 each row of the table is a $\langle valuation - valueset\rangle$ pair. Let $T = \{\langle valuation_1 - valueset_1\rangle, \langle valuation_2 - valueset_2\rangle, \cdots \langle valuation_n - valueset_n\rangle\}$. Thus in Figure 6.1, T is the entire table and each element of T is a row of the table. Let T' be the corresponding set $\{valueset_1, valueset_2, \cdots, valueset_n\}$. Let T'' be the cartesian product of the sets in T'. Each element e_i of $T'' = \{e_1, e_2, \cdots, e_m\}$, then, is a set of $\langle value, path, Interpretation\rangle$ triplets. $max^3(T)$ is then defined as

$$\begin{aligned} max^3(T) &= \{\langle value_r, path_r, I_r\rangle \\ &= max^2(e_i) \mid e_i \in T'', value_r > 0, I_r \neq \Rightarrow\Leftarrow\} \end{aligned}$$

Thus, $max^3(T)$ is the collection of results of max^2 applied to each element of T'' where the combined interpretation is consistent and the aggregate value is greater than 0. The cartesian product is taken to make sure that we consider all possible ways in which a map can be generated in the FODD.

The example in Figure 6.1 shows the result of applying max^3 to the elements in the table. Although there are $2 \times 3 \times 3 \times 2 = 36$ possible combinations of valuation behaviors (and hence 36 elements in T''), only 3 of these combinations result in a consistent combined interpretation and positive value. For example, under the given DPO, $max^2(\{\langle 10,\{1t\},\{p(a)\}\rangle, \langle 10,\{1t\},\{p(a)\}\rangle, \langle 10,\{1f2t\},\{p(a),\neg p(b)\}\rangle, \langle 10,\{1t\},\{p(b)\}\rangle\})$ $= \langle 10,\{1t\},\{p(a), p(b), \neg p(b)\}\rangle$ is omitted from the result of $max^3(T)$ because the combined abstract interpretation is inconsistent. Aggregations resulting in 0 value are ignored because 0, being the smallest obtainable value, is uninteresting under the max aggregation semantics. Observe that in this example, the path $\{1t\}$ is the only instrumental path. Intuitively this implies that the target of any edge not on this path (for instance edge $1f$) can be set to 0 without changing the map. The resulting FODD is shown on the right.

This process is formalised in procedures 6 and 7.

Procedure 6 *R12(B)*

```
1. Fix a DPO $PL$.
2. Invent as many new objects as the number of variables in $B$. Let $O$ be the set of these new objects.
3. Let $U$ be the set of all possible valuations of the variables in $B$ over $O$.
4. Let $S =$ Reduction-Aggregation($B, U, PL$) $=$ $\{\langle value_1, path_1, I_1\rangle, \langle value_2, path_2, I_2\rangle, \cdots \langle value_n, path_n, I_n\rangle\}$.
5. Let $E' = \{e \mid e$ is an edge in path $path_i$ and $\{leaf(path_i), path_i, I\} \in S$ for some $I\}$. Thus $E'$ is the set of all edges that appear on any path exposed in any triplet in the set $S$.
6. Define $E = B_E - E'$, where $B_E$ is the set of all edges in $B$.
7. $\forall$ edges $e \in E$, set target($e$) in $B$ to $0$ to produce FODD $B'$.
8. return $B'$.
```

Procedure 7 *Reduction-Aggregation(B, U, PL)*

```
1. Let $Val = \{\}$
2. Do for every valuation $\zeta \in U$
    (a) $valueset$ = getValue($\zeta$, $\{\}$, $\{\}$, $B_{root}$)
    (b) Add the entry $[\zeta, valueset]$ to $Val$
3. Let $T = max^3(Val)$ under $PL$.
4. return $T$
```

6.1.2 Proof of Correctness

This section shows that the R12 procedure removes exactly the right edges in its input FODD. We show that our procedure identifies the set of edges on instrumental paths and that all other edges can be removed, thus showing both soundness and completeness. We follow with the technical details.

Lemma 16 *If a path p_i in FODD B is instrumental under DPO PL, and reaches a non-zero leaf, then $\exists$ I_o such that $\{$leaf(p_i), p_i, $I_o\} \in S$.*

Proof: If p_i is instrumental under PL then $\exists$ I, ζ, $Path_B(I, \zeta) = p_i$ and $\forall$ η, $Path_B(I, \eta)$ $= p_j$ implies $j \geq i$. Let O' be the set of objects in I that participate in ζ. Clearly $1 \leq |O'|$ $\leq |O|$. Let o_1 be an object in O'. Add $|O| - |O'|$ new objects to O' to make the sets O and O' equal in size. Construct interpretation I' by first projecting I to include only the objects in O' and then defining truth values and predicates over the new objects to behave identical to o_1.

Since I' includes the relevant portion of I there is a valuation that traverses p_i under I'. Additionally, if $\exists\hat{\zeta}$, $Path_B(I', \hat{\zeta}) = p_j$, where $j < i$, we can construct valuation ζ by replacing the new objects in $\hat{\zeta}$ by o_1 so that $Path_B(I, \zeta) = p_j$. However, we know that $\forall$ η, $Path_B(I, \eta) = p_j$ implies $j \geq i$. Therefore we conclude that $\forall$ η, $Path_B(I', \eta) = p_j$ implies $j \geq i$.

Let U now, be the set of all valuations of the variables in B over O'. Let $I_o =$ $\bigcup_{\eta \in U} (PF(Path_B(I', \eta)))\eta$. That is, I_o includes all the atoms of I' that participate in traversing paths in B for all $\eta \in U$. By construction the corresponding parts $(PF(Path_B(I',$ $\eta)))\eta$ will be included in the $valueset$ returned by the getValue procedure. Clearly $I_o \subseteq I'$. Therefore if I' is consistent then so is I_o. By the definition of max^3, $S = max^3(Val)$ under PL must contain $\{$leaf(p_i), p_i, $I_o\}$ when leaf(p_i) is non-zero. ■

Lemma 17 *If there exists an instrumental path under PL that crosses edge e in B and reaches a non-zero leaf, then $e \in E'$.*

Proof: If there is an instrumental path $p_i \in PL$ that crosses edge e and reaches a non-zero leaf, then by Lemma 16 $\exists$ I_o such that $\{$leaf(p_i), p_i, $I_o\} \in S$. By definition of E', $e \in E'$. ■

Theorem 8 (soundness) *If FODD B' is the output of R12(B) for any FODD B, then $\forall$ interpretations I, MAP$_B(I) =$ MAP$_{B'}(I)$.*

Proof: By the definition of R12, the only difference between B and B' is that some edges that pointed to sub-FODDs in B, point to the 0 leaf in B'. These are the edges in the set E at the end of the R12 procedure. Therefore any valuation crossing these edges achieves a value of 0 in B' but could have achieved more value in B under the same interpretation. Valuations not crossing these edges will achieve the same value in B' as they did in B. Therefore for any interpretation I and valuation ζ, MAP$_B(I, \zeta) \geq$ MAP$_{B'}(I, \zeta)$ and hence MAP$_B(I) \geq$ MAP$_{B'}(I)$.

Fix any interpretation I and $v =$ MAP$_B(I)$. Let ζ be a valuation such that MAP$_B(I, \zeta)$ $= v$. If there is more than one ζ that gives value v, we choose one whose path p_j has the least index in PL. By definition, p_j is instrumental and by lemma 17, either leaf(p_j) $= 0$ or none of the edges of p_j are removed by R12. In both cases, MAP$_{B'}(I, \zeta) = v =$ MAP$_B(I)$. By the definition of the max aggregation semantics, MAP$_{B'}(I) \geq$ MAP$_{B'}(I, \zeta)$ and therefore MAP$_{B'}(I) \geq$ MAP$_B(I)$. ■

Theorem 9 (completeness) *If no path crossing edge e and reaching a non-zero leaf in B is instrumental under PL, then R12 removes e.*

Proof: By definition the set of all edges in B is partitioned into sets E and E'. Now, if $e \in E'$, then $\exists$ path $p_i \in PL$ and interpretation I_o such that e is an edge on p_i, leaf(p_i) is non-zero and $\{$leaf(p_i), p_i, $I_o\} \in S$. The existence of $\{$leaf(p_i), p_i, $I_o\}$ in S implies that under I_o, $\exists$ valuation $\zeta \in U$ such that $Path_B(I_o, \zeta) = p_i$ and $\forall$ $\eta \in U$, $Path_B(I_o, \eta) =$ p_j implies $j \geq i$. Therefore p_i is instrumental. Therefore all edges in E' belong to some instrumental path. This implies that e from the statement of the theorem is not in E' and therefore it is removed by R12. ■

Implementing R12 will not be practical for FODDs with more than a small number of variables because it involves enumeration of all possible valuations. On the other hand, previous reduction operators rely on theorem proving over single path formulas or edge implications. There are cases where such reduction operators fail to reduce a diagram but R12 is successful. Figure 6.2 shows an example where R12 succeeds but previous

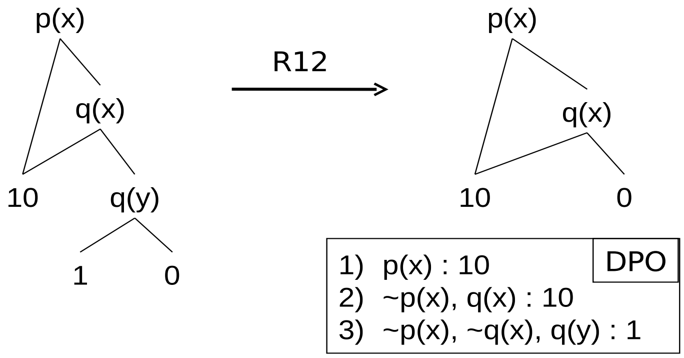


Figure 6.2: Example where R12 can reduce the diagram but previous reductions fail

reductions fail. Notice that there are 2 paths reaching the 10 leaf in the left FODD. In this diagram, whenever a valuation reaches the 1 leaf there is another valuation that reaches the 10 leaf through one of the 2 paths. However, neither of the path formulas are individually implied by the formula for the path reaching the 1 leaf. Similarly neither of the edge formulas for the edges terminating in the 10 leaf are implied by the edge formula for the edge terminating in the 1 leaf. R12, on the other hand, relies on model checking and is able to reduce the FODD on the left to the FODD on the right.

Above we gave soundness and completeness properties for R12. However, the completeness result falls short of providing a normal form because it relies on a DPO to define which parts of a diagram may be reduced when there are mutual implication relations. Therefore the same semantic function may have different minimal representations. However, the completeness guarantees are much stronger than those of previous reductions. In Chapter 3 we discussed normal form for FODDs. Examples of FODDs given there (using the length 2 path construction) show that for normal form we may need some syntactic manipulation of diagrams. Therefore going beyond the completeness shown in this chapter may be hard or expensive to compute. As a final comment, note that R12 is distinguished from previous reductions by the fact that it employs the aggregation function of the FODD itself as its main subroutine. This fact is useful when we generalize FODDs to allow different aggregation functions in Chapter 8.

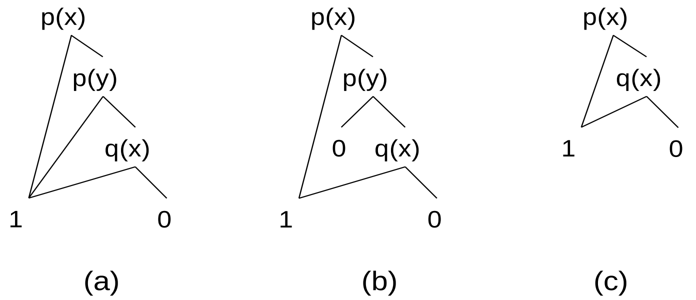


Figure 6.3: Example of $R12_{edge}$ and $R12_{node}$. Given the training set $\{\{p(1)$,$q(1)\}$, $\{\neg p(1)$,$q(1)\}\}$, $R12_{edge}$ reduces the diagram (a) to produce diagram (b). Given the training set $\{\{p(1)$,$q(1)\}$,$\{\neg p(1)$,$q(1)\}$,$\{\neg p(1)$,$\neg q(1)\}\}$ $R12_{node}$ removes the redundant node from (b) to produce (c).

6.2 Practical Model-Checking Reductions

As mentioned in the previous section, R12 avoids enumerating all possible interpretations through bookkeeping but the complexity of R12 is too high requiring an enumeration of n^n valuations to a hypothetical interpretation for a diagram with n variables. In the following we present two simple heuristic variants of R12 that allow us to remove edges as well as nodes in FODDs. We assume a given set of “focus interpretations” that together capture all important variation in the state space. Note that we do not assume all interpretations of interest are given but instead we assume that if an important condition exists for the domain then this condition is realized in at least one of the given interpretations. This is a much weaker condition. The result is an efficient variant of R12 and of an extension of R12 for node removal. As a result we lose soundness and may over-prune a diagram if the set of given states is not sufficiently rich. On the other hand if the set does capture all important distinctions in the domain we get efficient, sound and complete reductions.

6.2.1 Edge Removal by Model Checking

For edge removal we want to determine when an edge pointing to a sub-diagram can be replaced with a zero leaf. Recall that MAP$_B(I) = max_\zeta$MAP$_B(I)$. Therefore the values provided by the non maximizing ζ’s can be reduced without changing the final results. As above, an edge is instrumental if it participates in a path that gives the final value on some interpretation. In the following this is approximated by being instrumental on the given examples. This idea can be easily implemented as follows:

Procedure 8 $R12_{edge}$
Input: FODD B, Sample E
Output: Reduced FODD B'

```
1. Generate a DPO $P$ for $B$.
2. $I = \{\}$
3. For each example $e$ in $E$,
   For $i = 1$ to $|P|$,
   if $p_i$ subsumes $e$, then $I = I \cup$ edges($p_i$) ; break
4. For each edge $e'$ in $B$ such that $e' \notin I$, set target($e'$) $= 0$
```

Clearly every path identified as instrumental and added to I is instrumental. Therefore we prune all unnecessary edges. On the other hand if the example set is too poor, we may over prune the FODD. Notice that as long as the given examples satisfy domain constraints we will automatically prune any paths violating such constraints without the need to employ complex background knowledge. Similarly any implied relation among predicates is automatically and implicitly used in the reduction. This is a significant practical feature of the new reductions and already provides an advantage over the theorem proving reductions. In practice this reduction is also much faster than theorem proving reductions of similar scope.

On the other hand, the quality of the reduction is strongly dependent on the quality of the example set. The set has to be representative so that important structure is not reduced from the diagram. At the same time we want fewer examples to improve efficiency. Figure 6.3 shows an example of $R12_{edge}$. Given a training set $\{\{p(1), q(1)\}, \{\neg p(1), q(1)\}$, where the domain contains only one object $\{1\}$, the FODD in Figure 6.3(a) is reduced by $R12_{edge}$ to produce the FODD in Figure 6.3(b). When the DPO is constructed so as to give precedence to shorter paths, the path $p(x)$ is deemed instrumental by $R12_{edge}$ because of example $\{p(1), q(1)\}$ and the path $\neg p(x), \neg p(y), q(x)$ is deemed instrumental because of the example $\{\neg p(1), q(1)\}$. In fact this is the smallest training set that removes all redundant edges from the diagram without over pruning it.

6.2.2 Node Removal with Model Checking

While edge removal is important it does not handle a common type of redundancy that arises often when FODDs are composed. For instance, in the SDP algorithm we add or multiply functions with similar structure that are standardized apart (step 2 of the relational VI algorithm described in Chapter 2). Often we have an irrelevant node above an important portion of the diagram. We cannot remove the edge from that node because it will cut off the important sub-diagram. Instead what we need is a reduction that can skip the irrelevant node. This is similar to the issue handled by R11. We use this idea for nodes where one child is zero and the other is a diagram. The question is whether connecting the node's parents directly to the non-zero child will change MAP$_B(I) = max_\zeta$MAP$_B(I)$. The only way this can happen is if a valuation ζ that previously went to the zero child is now directed to a non-zero leaf which is greater than the previous maximum. As above, this condition is easy to check directly on the given set of examples.

Procedure 9 $R12_{node}$
Input: FODD B, Sample E, Set of Candidate nodes C
Output: Reduced FODD B'

```
1. Generate a DPO $P$ for $B$.
2. For each node $n \in C$ do the following:
    (a) Remove node $n$ from $B$ by connecting the parents of $n$ directly to the non-zero child of $n$ to produce FODD $B_{-n}$.
    (b) Generate a DPO $P_{-n}$ for $B_{-n}$.
    (c) Set $keep.node = 0$.
    (d) For each example $e$ in $E$, do the following:
          i. For $i = 1$ to $|P|$,
             if $p_i$ subsumes $e$, then set value($e$) = leaf($p_i$) ; break
         ii. For $i = 1$ to $|P_{-n}|$,
             if $p_i$ subsumes $e$, then set newvalue($e$) = leaf($p_i$) ; break
        iii. If newvalue($e$) $>$ value($e$), set $keep.node = 1$; break
    (e) If $keep.node == 0$, set $B = B_{-n}$
```

As above, any node with $keep.node = 1$ must be kept otherwise the value corresponding to some example will change. Therefore we prune as much as is allowed by the example set. Again, if the example set is not rich enough, we may over prune the FODD. For example we saw that the example set $\{\{p(1), q(1)\}, \{\neg p(1), q(1)\}\}$ is sufficient to remove all edge redundancies while still maintain soundness and reduce it to the FODD in Figure 6.3(b). With the same example set, however, the R12-node reduction will remove the non-redundant node $q(x)$. This is because the value of neither example in the set changes by the removal of $q(x)$. For $q(x)$ to survive R12-node the example set must have an example like $\{\neg p(1), \neg q(1)\}$ demonstrating that the node is important. Adding $\{\neg p(1), \neg q(1)\}$ to the example set reduces the diagram in Figure 6.3(b) to the diagram in Figure 6.3(c), which is the smallest FODD representation of the function $p(x) \vee q(x)$.

6.3 Discussion

In this chapter we introduced the reduction operator R12 based on model-checking. Although R12 has theoretical properties that are superior to the theorem proving reductions, it is not a practical algorithm. We presented practical versions of R12, $R12_{edge}$ and $R12_{node}$. The efficiency of $R12_{edge}$ and $R12_{node}$ is significantly better than that of theorem proving reductions. With a good set of training examples, these reductions preserve important structure in the diagram. The quality of the training set has to be defined relative to a specific application. In the next Chapter we continue with Decision Theoretic Planning as our application to demonstrate how a “good” training set can be generated for this application. At the same time we will demonstrate the superior efficiency of $R12_{edge}$ and $R12_{node}$ over other reductions.

Chapter 7

Self-Taught Decision Theoretic Planning with FODDs

Inspired by the model-checking reductions developed in the previous chapter, we introduce a new paradigm for planning by learning: the planner is given a model of the world *and* a small set of states of interest, but *no* indication of optimal actions in any states. This paradigm is motivated by the observation that many state descriptions generated when solving one planning problem contain basic information that is important for solving other planning problems. Similar to implicit imitation for reinforcement learning of Price and Boutilier (2003), the additional information can help focus the planner on regions of the state space that are of interest and lead to improved performance. Naturally we validate and demonstrate the idea in the context of the FODD-PLANNER but the same technique is applicable to any SDP algorithm that requires logical simplification.

Focused planning as outlined above requires a set of training examples. We show that such training examples can be constructed on the fly from a description of the planning problem. Thus we can bootstrap our planner to get a self-taught planning system. We propose several such approaches, based on backward random walks from goal states enhanced with specific restrictions to ensure coverage of a rich set of states with a small sample. To recap, the idea is that given a description of the domain, we first generate the focus states automatically and then run SDP using FODDs as before but using model-checking reductions with the focus states as training examples instead of theorem proving reductions.

There have been other approaches where training examples have been used to generate models for solving planning problems (Fern et al., 2006; Gretton & Thiebaux, 2004). However, our approach differs from these approaches in that the training set does not need information about optimal actions or values along with the examples.

We implemented the model-checking reductions and the example generation routines in the FODD-PLANNER and applied it to several domains. In this chapter we provide an extensive evaluation of the ideas described above as well as several other system related issues, using the experiments to investigate and demonstrate the contribution of different aspects of the system.

The rest of the chapter is organized as follows. Section 7.1 describes ideas on generating examples to focus the planner. Section 7.2 provides the experimental evaluation in which we explore the merits of different methods to generate the training examples, provide "learning curves", explore speedup mechanisms, and give the experimental results on IPC problems.

7.1 Bootstrapping: Example Generation

The key to effective employment of $R12_{edge}$ and $R12_{node}$ is to provide these operators with a rich set of interesting examples. In the case of planning problems, the examples of interest are states visited during execution of the solution to a planning problem. Such states can be generated in a variety of ways. One potential method starts from a random state and runs episodes of simulated random walks through the state space. Another potential method employs a planner to solve a few sample planning problems and collect the states on the solution paths into the set of examples.

The methods most relevant to Decision Theoretic Planning start from a set of typical goal states and regress over ground actions to generate states from which the goal state is reachable. Regression from a ground state s over action a is possible only when a can achieve s from some state s'. Using STRIPS notation, this is easily verified by checking if s is subsumed by *PRECONDITIONS(a)* $+$ *ADD-LIST(a)* $-$ *DELETE-LIST(a)*. Such an s' is then generated by simply adding *DELETE-LIST(a)* to s and removing *ADD-LIST(a)* from s. This method of example generation is particularly suitable for SDP based systems like

FODD-PLANNER because the same states are assigned new values by VI. In the following, we develop two variants of this approach both of which take a seed state s and regress from it.

Instance Regression (IR): We iteratively generate all possible states up to a certain specified depth using a BFS procedure. Regression over states at depth d produce states at depth $d+1$ in the d^{th} iteration. The depth parameter can be set to the same value as the number of iterations of VI run by FODD-PLANNER because states from deeper levels are not relevant to the value function. Therefore there is no need for parameter selection of the depth parameter. However, this method could generate a large example set eliminating the advantage of model-checking reductions. To mitigate this effect we introduce the following pruning techniques.

(1) Limit IR to use only those actions that bring about certain literals in the state. These literals are the preconditions of the previous action that generated this state through regression. In particular, if state s was regressed over action a' to produce state s' in the i^{th} iteration, then the preconditions of a' (which must be true in s') are maintained as special literals in the description of s'. When regressing over s' in iteration $i+1$, only those actions that bring about these special literals are considered. Thus we try to generate states that are further away from the goal.

(2) Identify and mark sub-goals so that they are not achieved more than once. For example, suppose $s_p \subset s$ are the special literals in s and as above we regress from s using a'. Mark literals in $s_p \cap$ *ADD-LIST(a')*. Now, in the $i+1^{th}$ iteration, when regressing from s', only those actions that generate states not containing the marked literals are considered. Considering the sequence of actions generated as a plan, this heuristic avoids re-achieving the same goal literal by the plan. Although incomplete, this heuristic is effective in limiting the number of states generated.

(3) Regress states over a composition of k actions instead of a single action. The parameter k is independent and thus requires parameter selection. For example suppose action a is a composition of actions a'' and a'. Now, if s' regressed over a'' generates state s'', then s regressed over a generates s''. We add states s, s' and s'' to the set of examples. Regression continues from state s''. This technique avoids imposing the special and marked literal restrictions on every action and imposes them directly on compositions of actions, thereby

allowing more freedom to search the state space in cases where the previous techniques are too restrictive. In our experiments we used this option with $k = 2$ whenever insufficient data was generated by setting $k = 1$ (as shown by planning experiments).

Backward Random Walk (BRW): Instead of generating all states by iterative regression, as in IR, we run episodes of random walks backwards from the goal (sampling actions uniformly) without any of the above restrictions. This provides a varied set of states including states that are off the solution path for typical planning problems. Here the length and number of episodes require parameter selection. In our experiments we choose these arbitrarily so as to add examples but not increase set size to be too large.

In practice we need a mix of examples generated by IR and BRW. To see why, let V be a value function that solves planning problem p optimally and S be a set of all possible states along any optimal solution path of p. Then V reduced by $R12_{edge}$ against S is guaranteed to solve p optimally but V reduced by $R12_{node}$ against S is not. As illustrated above, $R12_{node}$ requires examples that demonstrate that the diagram will erroneously gain value on removal of important nodes. Hence states off the solution path of p might be required for V to be retained by $R12_{node}$. The techniques with IR, however, are designed to generate only states along solution paths. Therefore in the example set we include all the states generated by IR and add states generates by BRW to yield a mixed set.

7.2 Experiments on Planning Domains

In this section we present the results of our experiments on domains discussed in chapter 5, namely tireworld and blocksworld from IPC 2006, and the boxworld domain from IPC 2008. We left out the domains fileworld and logistics because we were able to achieve convergence of the policy in these domains and hence there is little scope for improvement. Our intention here is to investigate

> **(Q1)** what are the contributions of the different parameters (number of training examples, number of iterations of VI, types of reductions, goal ordering heuristic) of the self-taught model-checking system, and **(Q2)** whether our self-taught model-checking system can effectively speed up decision theoretic

planning while matching performance with the theorem proving system and other state of the art systems in stochastic planning.

To this aim, we generated a value function for each domain by running FODD-PLANNER in three different configurations based on the method used to reduce FODDs.

1. **FODD-PL:** All theorem proving reductions only. This is the same system we presented in chapter 5. All configurations or parameter settings of FODD-PL in these experiments are exactly the same as the ones used in the experiments in chapter 5.
2. **ST-FODD-ER:** R12 edge removal reduction and a theorem proving node removal reduction R11.
3. **ST-FODD-ERNR:** R12 edge and node removal reductions only.

In the following, we present a comparison study of the three methods over the same metrics we used in the experiments in chapter 5, i.e. their ability to solve planning problems from IPC measured in terms of coverage, average plan length, cpu time required to generate the value function (or policy), and in the case the boxworld domain, the average reward achieved. Average reward is irrelevant for blocksworld and tireworld because there are no action costs in these domains. We generated examples by mixing states generated by the IR and BRW methods as explained above. Again, we set a limit of 200 steps on the plan length when solving IPC problems. Plans that ran for more than 200 steps were counted as failed.

Overall we observe that self-taught planning (Methods ST-FODD-ER and ST-FODD-ERNR) generates the value function much faster than FODD-PL while maintaining the same level of performance. As in Chapter 5, plan execution time standards of IPC were not met for boxworld problems and some hard problems of blocksworld due to slow policy evaluation in the online component. Further research on this topic is needed to optimize for a faster execution module.

7.2.1 Timeout Mechanism

When building the model or executing a plan, the most expensive operation in FODD-PLANNER is subsumption (that is given a state and a path formula, solving the matching

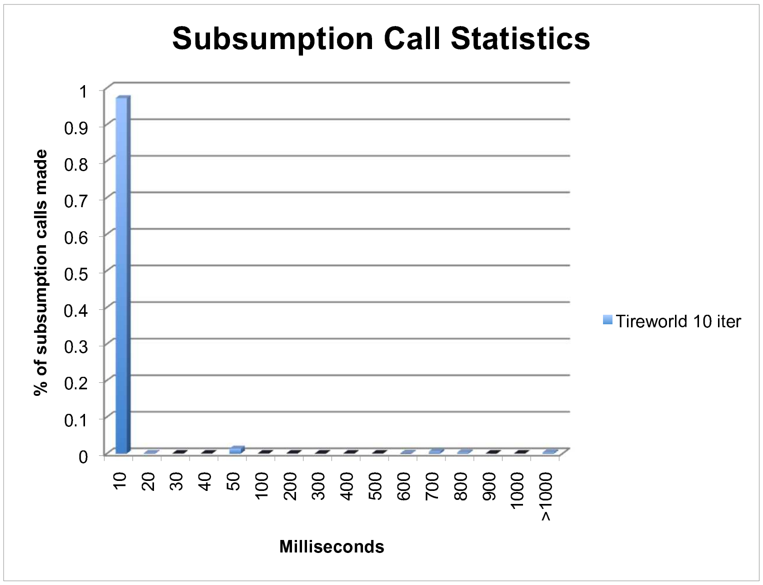


Figure 7.1: Subsumption Call Statistics: More than 99 percent of the call run under 50 milliseconds

Ex	50	100	150	200	250
Calls	145298	240157	320509	689286	852904

Table 7.1: Number of subsumption calls made during VI using ST-FODD-ER with given number of examples (Ex)

problem to decide whether the formula is applicable in the state). The basic observation is that because we switched from theorem proving to model-checking the complexity of the system is heavily affected by the cost of subsumption tests. As has been well documented in the literature (Maloberti & Sebag, 2004) subsumption problems show a phase transition phenomenon and, while most problems are easy, certain types of problems have a very high cost. In our context it turns out that very few subsumption tests are of this class and therefore we can get significant speedup by detecting and stopping these early, with little or no difference in coverage for the planning problem. Table 7.1 shows the number of subsumption calls made by FODD-PLANNER using ST-FODD-ER with varying number of training examples when building a model for 7 iterations of the tireworld domain. Since the number of calls is large, unless the amount of time per call is somehow controlled, we could be faced with prohibitively large run times.

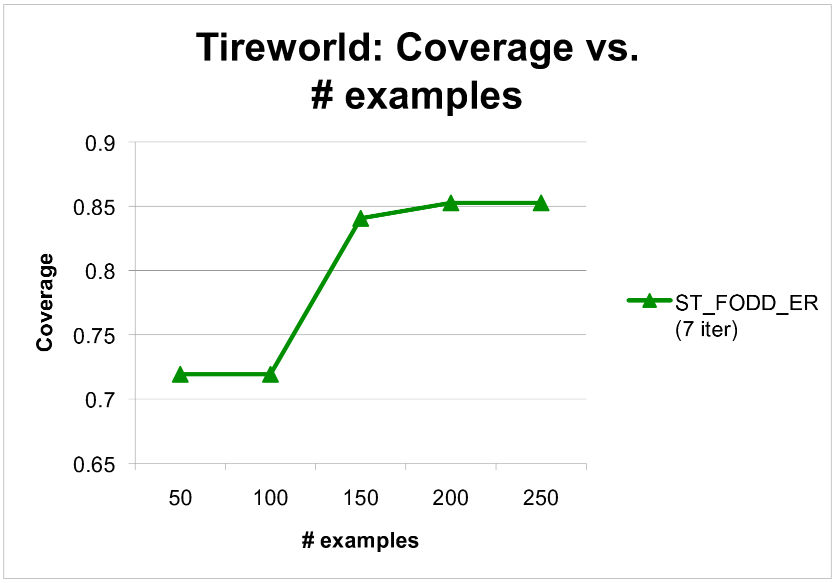


Figure 7.2: Tireworld Learning Curve: Average Percentage of Problems solved by ST-FODD-ER vs. Number of examples in the training set

In order to alleviate this cost, we utilize a timeout mechanism within the subsumption routine. The subsumption simply fails if it runs beyond a specified time limit. To decide on a setting for the time limit parameter, we generated histograms of run times of single subsumption tests. Figure 7.1 illustrates the percentage of subsumption calls made against the cpu time. We observe here, that more than 99 percent of the calls that are made run within 50 milliseconds. The most expensive call took more than 35 minutes. Therefore a time limit parameter to 1 second would cause only a negligible percentage of calls to time out yet eliminating the significantly more expensive subsumption calls. In initial experiments we were able to verify that the run time was greatly improved by setting a time limit of 1 second while keeping the same level of performance in solving planning problems. Additional timing experiments showed that not much more can be gained by fine tuning the timeout threshold. Therefore this heuristic provides a robust mechanism to control run time increase due to subsumption tests.

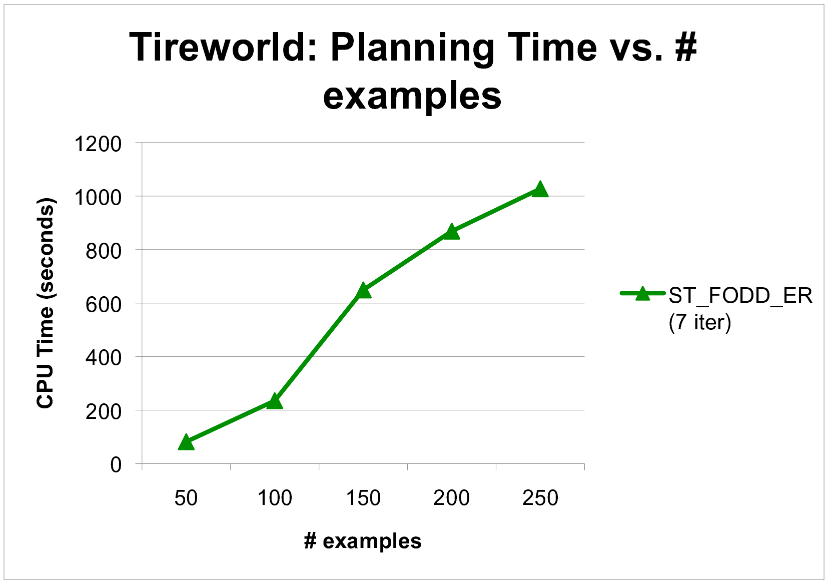


Figure 7.3: Tireworld: Planning time in cpu seconds by ST-FODD-ER vs. Number of examples in the training set

7.2.2 (Q1) System Characteristics

Since the model checking reductions remove all parts of the FODD that are not instrumental in determining the map for the set of training examples, similar to standard machine learning, we would expect the size of the training set to have an effect on the level of approximation in the value function and hence on planning accuracy. Smaller training sets would cause important structures in the value function to be eliminated and consequently show poorer performance than larger training sets. In order to understand and illustrate this, we experimented on building the value function for 7 iterations on the tireworld domain using ST-FODD-ER varying the example set size. Figure 7.2 shows the learning curve produced. We increased the example set size from 50 to 250 with each set containing all examples from the previous set and 50 additional random examples. The Y-axis shows average coverage over the 15 IPC 2006 problems. We see that the performance converges before reaching a 100%. This is because some of the problems are designed with action failures and deadlocks so that achieving a coverage of 100% when averaged over 30 rounds is statistically very unlikely. For the same setting, Figure 7.3 shows a linear increase of run time against the number of training examples. Figure 7.4 shows the run time against the

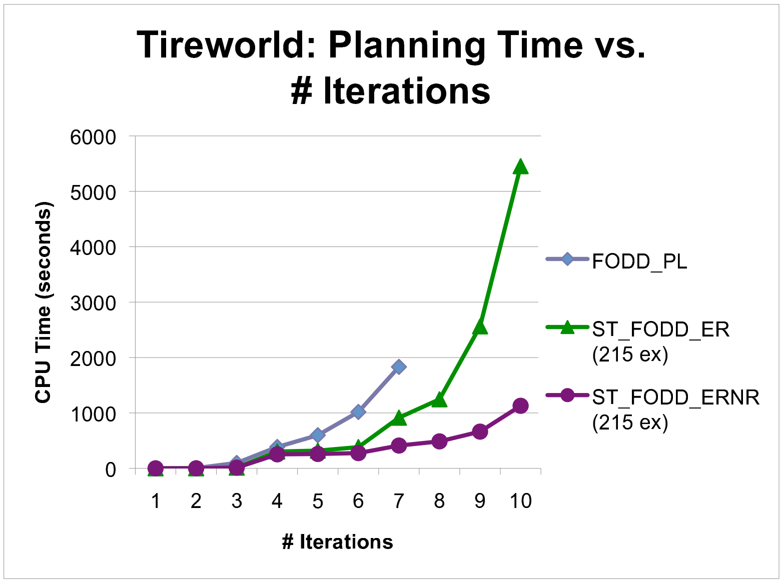


Figure 7.4: Tireworld: Planning time in cpu seconds vs. Number of iterations of VI

M	PL	ER	ERNR	ER	ERNR
I	7	7	7	10	10
C	1831.69	914.428	542.464	5457.76	1129.02

Table 7.2: Tireworld: Planning time taken in CPU seconds (C) by methods (M) FODD-PL (PL), ST-FODD-ER (ER) and ST-FODD-ERNR (ERNR) run for various number of iterations (I)

number of iterations of VI for the 3 methods. For ST-FODD-ER and ST-FODD-ERNR, a fixed training set of 215 examples was used. Clearly ST-FODD-ERNR is more efficient than ST-FODD-ER, which in turn is more efficient than FODD-PL. This allows us to run more iterations of VI with the system based in model-checking reductions.

7.2.3 (Q2) Tireworld

Training examples for this domain were generated by running IR with the d parameter set to 10 and the k parameter set to 1 and running BRW for 10 episodes of length 20 each. The seed state for both methods was designed so as to have a map of 10 locations connected linearly. Each location connects to 2 neighbors by a two way road except the locations at the extremes which connect to only one location. The vehicle was placed in the first

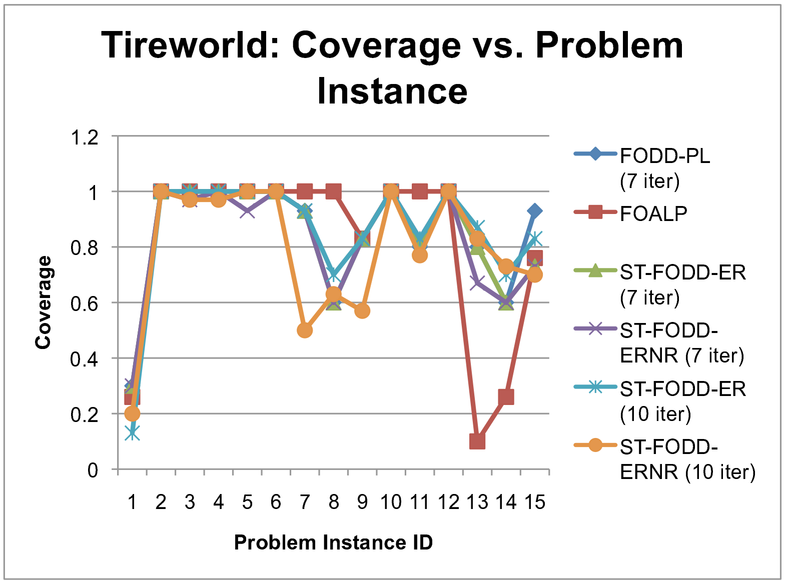


Figure 7.5: Tireworld Coverage Results

location. This was chosen heuristically as it seemed the simplest configuration relevant for tireworld.

Following the experimental procedure in chapter 5, we apply the non-std-apart approximation after the 3^{rd} iteration of VI. Figures 7.5, 7.6 and Table 7.2 show the comparison of the 3 methods. We observe here that the model checking methods ST-FODD-ER and ST-FODD-ERNR generate a value function much faster than FODD-PL while keeping the same level of performance on the IPC problems. It is also important to note that these results of coverage and plan length are competitive with the performance of FOALP (Sanner & Boutilier, 2009), one of the top ranking systems from IPC 2006. Results for plan length show comparable results to FODD-PL which is slightly better than other systems.

Since the model checking reductions allow faster planning, we can run more iterations of VI to generate a deeper value function that can solve harder planning problems. Figure 7.7 shows a simple planning problem where the value function generated by 7 iterations of the FODD-PLANNER using FODD-PL fails but one generated by 10 iterations of FODD-PLANNER using ST-FODD-ER succeeds. The vehicle is at location a and has to get to location j. Spare tires are available in all locations. The problem is designed such that at every step along the way, there are 4 wrong actions that lead to dead end states and one

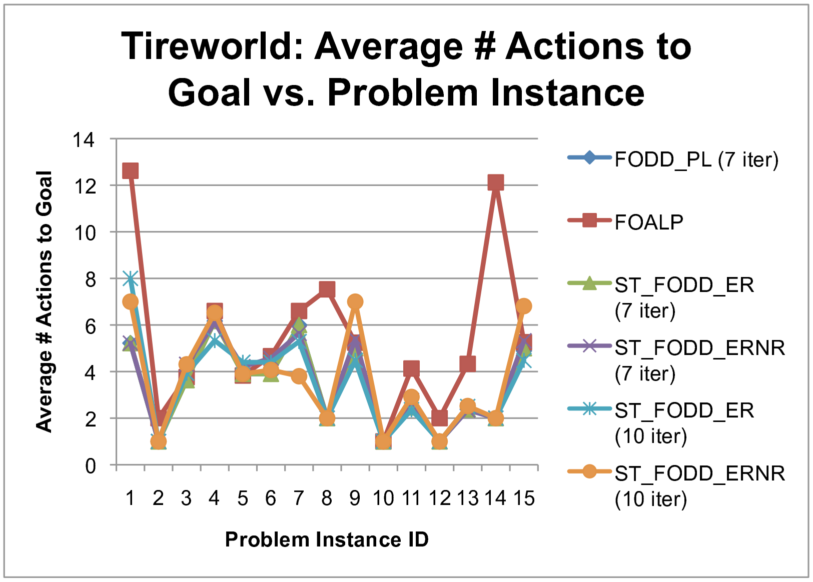


Figure 7.6: Tireworld Plan Length Results

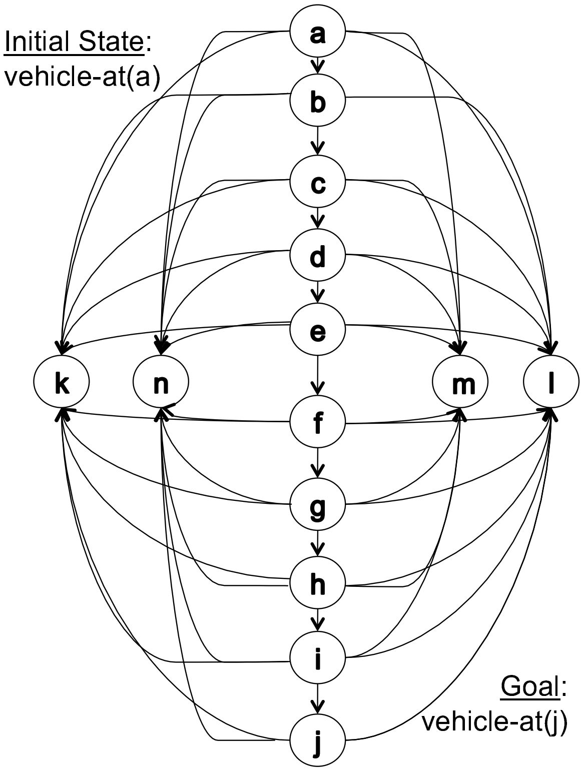


Figure 7.7: Tireworld Challenge Problem

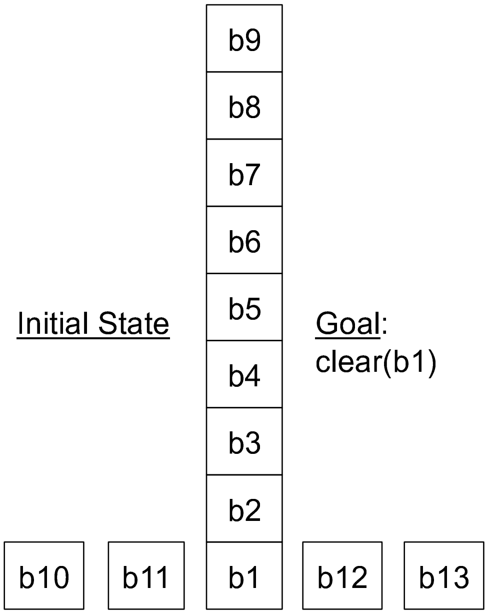


Figure 7.8: Blocksworld Challenge Problem

correct action that makes progress. Thus one wrong action will cause the plan to fail. So only a value function deep enough to assign a non-zero value to the starting state can solve it. We could not generate such a problem using the IPC problem generator for tireworld. The problem generator produces example problems by creating random location maps of given edge density and choosing random connected initial and goal locations. We observed that independently of the edge density parameter the initial and goal location appeared no farther than 5 steps from each other. Running 10 iterations with FODD-PL takes more than 2 days whereas with ST-FODD-ER takes about and hour and a half and with ST-FODD-ERNR takes about 16 minutes.

7.2.4 Blocksworld

As before, training examples for this domain were generated by both methods IR and BRW. For IR the d parameter was set to 15 and the k parameter was set to 2. For BRW we ran 10 episodes of length 20 each. For both methods, the seed examples for each goal (the 4 goals in blocksworld are *clear-block(a), on(a,b), on-table(a)* and *arm-empty*) was designed to be a state satisfying the single goal literal and, in addition, a tower including all

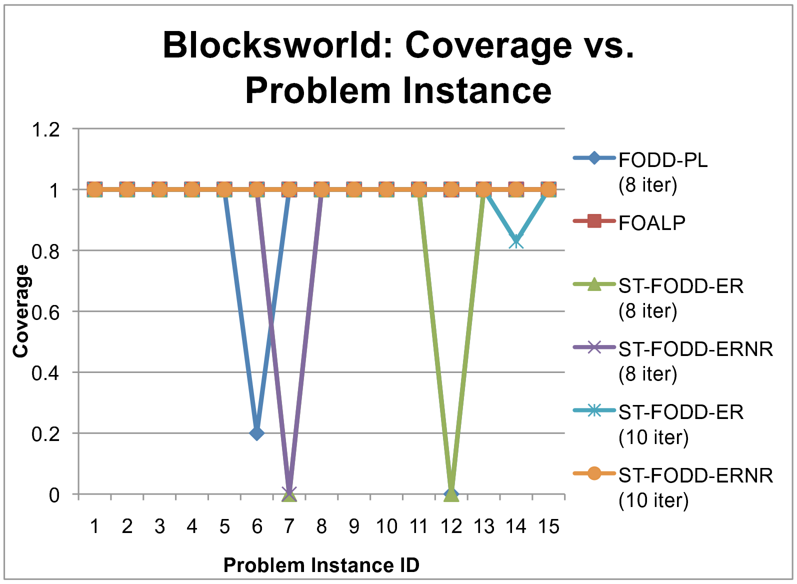


Figure 7.9: Blocksworld Coverage Results

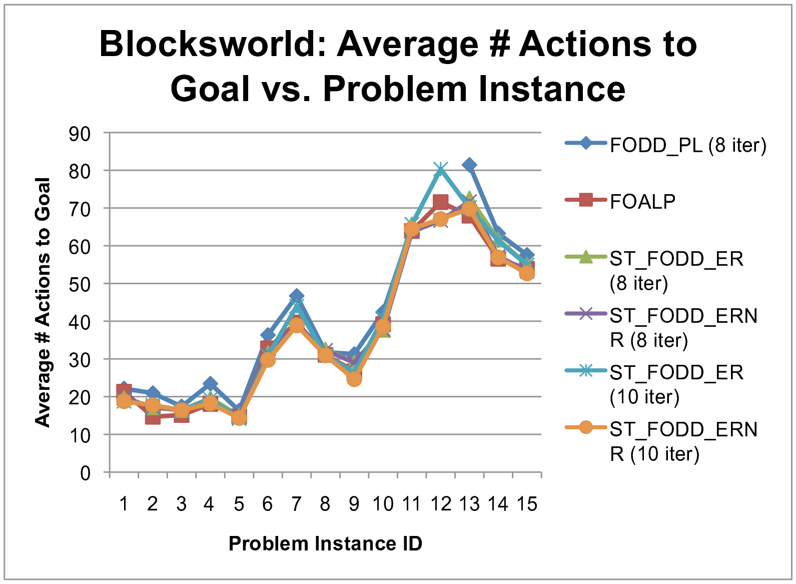


Figure 7.10: Blocksworld Plan Length Results

M	PL	ER	ERNR	ER	ERNR
I	8	8	8	10	10
C	12465.95	222.33	78.69	865.57	357.68

Table 7.3: Blocksworld: Planning time taken in CPU seconds (C) by methods (M) FODD-PL (PL), ST-FODD-ER (ER) and ST-FODD-ERNR (ERNR) run for various number of iterations (I)

other blocks. This was chosen heuristically because it seemed the simplest configuration relevant for blocksworld. Examples just from IR were sufficient to provide the results shown for ST-FODD-ER but examples from BRW were required for ST-FODD-ERNR to perform well. Table 7.3 and Figures 7.9 and 7.10 show the comparison of the 3 methods on the 15 IPC 2006 problems. We observe that for 8 iterations ST-FODD-ER and ST-FODD-ERNR achieve respectively 56 and 158 fold speedup in terms of planning time over FODD-PL while keeping the same level of performance on the IPC problems. ST-FODD-ERNR in particular performs better than the other two in terms of both coverage and plan length. Once again the results presented here are competitive in comparison with top ranking systems from IPC 2006. The results of FOALP are presented on the same graph for comparison. Plan execution times were very similar to the ones in chapter 5.

As before the IPC problems do not illustrate dramatic improvement in performance due to a larger number of iterations. But such problems are not rare and can be easily designed. For example Figure 7.8 shows a problem at which the value function generated by 8 iterations of FODD-PLANNER with FODD-PL achieves only 50% coverage but one generated by 15 iterations of FODD-PLANNER with ST-FODD-ERNR achieves full coverage. This shows that the ability of the model checking methods to run more iterations pays off when solving harder problems. This problem was designed so that a shallow value function would get lost in the action space (as there are many block to move) whereas a deep value function would be able to take the right actions (unstacking the tower) from the start state itself. Running 15 iterations using FODD-PL takes more than 3 days whereas with ST-FODD-ERNR takes less than 5 hours.

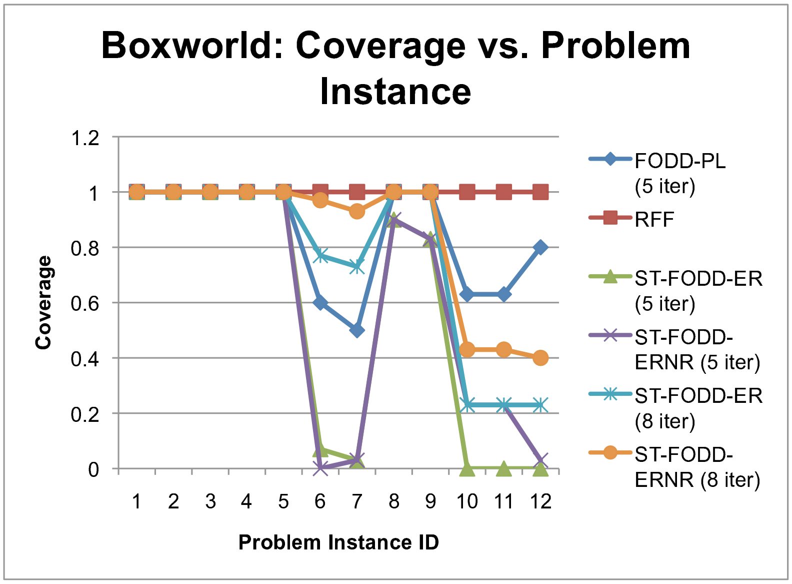


Figure 7.11: Boxworld Coverage Results

M	PL	ER	ERNR	ER	ERNR
I	5	5	5	8	8
C	3332.33	302.8	117.7	3352.93	618.85

Table 7.4: Boxworld: Planning time taken in CPU seconds (C) by methods (M) FODD-PL (PL), ST-FODD-ER (ER) and ST-FODD-ERNR (ERNR) run for various number of iterations (I)

7.2.5 Boxworld

Training examples for this domain were generated by IR and BRW. For IR we set the d parameter to 8 and the k parameter to 1. For BRW we ran 10 episodes of length 20 each. For both methods, the seed example was designed so as to have a map of 10 cities connected linearly. Each location connects to 2 neighbors by a two way road except the locations at the extremes which connect to only one location. 2 boxes, 2 trucks and a plane were was placed in the 4^{th} city from one extreme.

Table 7.4 and Figures 7.11, 7.12 and 7.13 show a comparison of the 3 methods on the 12 IPC 2008 problems. For comparison, the results of RFF are plotted on the same graph. ST-FODD-ER and ST-FODD-ERNR respectively achieve 11 and 28 fold speed up over FODD-PL at 5 iterations and enable running a larger number of iterations. Similar to the

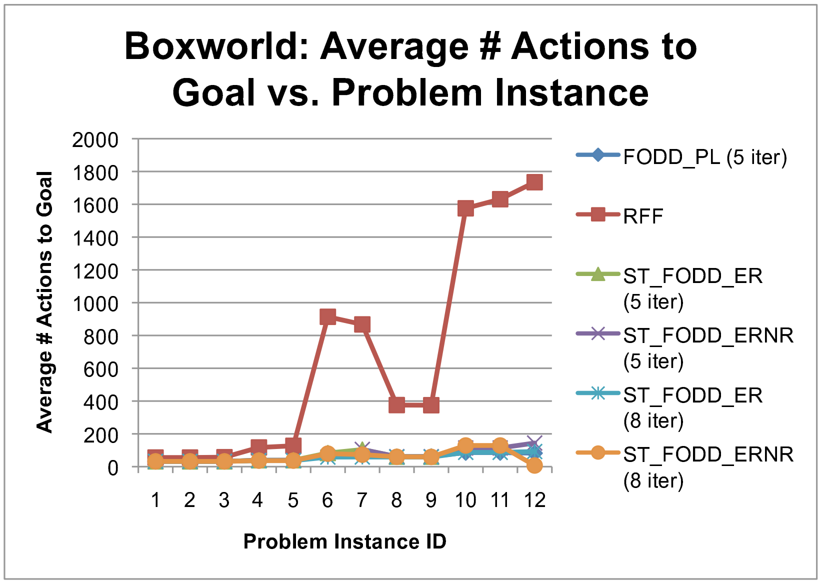


Figure 7.12: Boxworld Plan Length Results

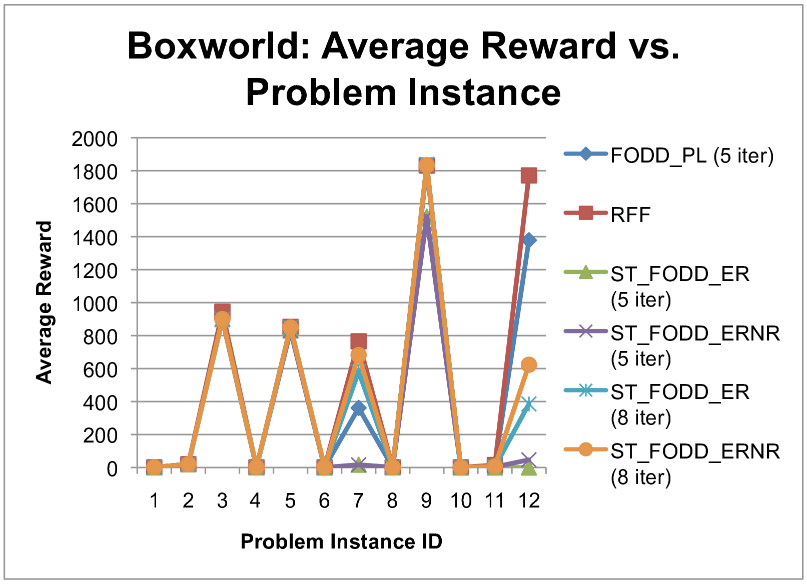


Figure 7.13: Boxworld Plan Length Results

experiments in chapter 5, we faced long execution times varying from 100 seconds per round on the easier problems to 1.5 hours per round on hard problems.

We observe again that ST-FODD-ER and ST-FODD-ERNR outperform FODD-PL in terms of planning time while maintaining the level of performance. None of the methods are able to outperform RFF in terms of coverage, especially for problems 10, 11 and 12. RFF achieves full coverage on all the 12 problems. However, as in Chapter 5, our performance is close to RFF in terms of accumulated reward and we perform consistently better in terms of plan length even on problems where we achieve full coverage.

7.3 Summary and Concluding Remarks

To summarize, our experiments demonstrate that the new paradigm of self-taught planning leads to a significant speedup of Symbolic Dynamic Programming through the use of a set of states of interest. Self-taught or provided by a mentor, the information gleaned from these states of interests is used to remove any complex specifications within the value function that are irrelevant to the given states, thereby focusing on the region of interest. In this light, the paradigm can also be described as unsupervised transfer learning for planning. Experiments on domains and planning problems from the International Planning Competitions suggest that this technique not only greatly improves the efficiency of the planning system, but also allows it to solve harder planning problems. The systems ST-FODD-ER and ST-FODD-ERNR show orders of magnitude improvement in efficiency over FODD-PL demonstrating the superior efficiency of model-checking reductions over theorem proving reductions.

Although all experiments and demonstrations have been in the context of FODDs and the FODD-PLANNER system, we believe that self-taught planning can also be applied to other SDP based solvers of RMDPs. Exploring this is an interesting direction for future work. Another promising idea is to acquire the training examples from multiple teachers each specializing in a separate (but possibly overlapping) part of the state space. In contrast with behavioral cloning (Morales & Sammut, 2004) where the performance typically degrades if the learner gets contradicting examples from multiple teachers, our approach handles this case in a natural way.

Chapter 8

Generalized First Order Decision Diagrams

FODDs have an inherent limitation in terms of representation power. FODDs (roughly speaking) represent existential statements but do not allow universal quantification. This can be a limitation for many tasks that require more expressive representations. This excludes some basic planning tasks. For example, a company that has to plan a recall of faulty products requires quantifier prefix $\exists\forall$ for the goal: there exists a depot such that all products are in the depot. Towards overcoming this limitation, this chapter introduces Generalized FODDs (GFODD), a novel FODD extension that allows for existential and universal quantification and other aggregators of value. We develop a theoretical framework for GFODDs by characterizing GFODD composition and introducing an extension of the model-checking reduction R12 for GFODDs for the quantifier settings $\exists^*\forall^*$. That is a finite number of $\exists$ quantifiers followed by a finite number of $\forall$ quantifiers. Finally we refer to the Decision Theoretic Planning application again by presenting a SDP based algorithm showing how GFODDs can be used to solve RMDPs with arbitrary quantification. This is a significant extension of the scope of the FODD approach to decision-theoretic planning. This also advances the theoretical understanding of probabilistic inference with large models.

The chapter is organized as follows. Section 8.1 introduces GFODDs and their composition operations. Section 8.2 extends the model-checking reduction operator to GFODDs with the quantifier setting $\exists^*\forall^*$. Finally Section 8.3 shows the utility of GFODDs for solving RMDPs.

8.1 Generalized FODDs: Syntax and semantics

The max aggregation of FODDs makes them sufficiently expressive to represent many planning problems of interest. However, since the max aggregation mirrors existential quantification over the variables of the FODD, many other functions over logical spaces cannot be represented by FODDs. These functions could be represented if the aggregation function was more complex. For example, employing a min aggregation instead of a max allows representation of functions where the variables are universally quantified. Similarly one can imagine using other aggregation functions like the sum, the $mean$, the $variance$ and also a complex mix of these aggregation operators. In this section and the next, we discuss the properties of such generalized FODDs and the operations that can be performed to manipulate them. We start by a formal definition of Generalized First Order Decision Diagrams.

Definition 5 *A Generalized First Order Decision Diagram (GFODD) is a 2-tuple $\langle V, B\rangle$, where, V is an ordered list of distinct variables each associated with its own aggregation operator. We call V the aggregation function for the GFODD. A variable cannot be aggregated in more than one way. Therefore V contains no repetitions. B is a FODD except that the leaves can be labeled by a special character d (for discard).*

8.1.1 Semantics of GFODDs

The semantics we choose for GFODDs is similar to the ones for FODDs except that the aggregation operation is now defined by $V = [(op^1_{v_1}), (op^2_{v_2}) \cdots (op^n_{v_n})]$. Here every v_i is a variable of the GFODD (ordered v_1 to v_n) and the corresponding op_i is the aggregation operator associated with variable v_i. Consider the set of all possible valuations over variables $v_1 \cdots v_n$ defined for the domain of interpretation I. Each valuation ζ is associated with a value MAP$_B(I, \zeta)$. We can now divide up these valuations into blocks. All valuations in a block have the same assignment of values to variables $v_1 \cdots v_{n-1}$ but they differ in the value of the variable v_n. We then "collapse" each block to a single valuation over variables $v_1 \cdots v_{n-1}$ by eliminating the variable v_n and replacing the set of associated values (MAP$_B(I, \zeta)$) by their aggregate value produced by applying op^n to the set. Any discard values in the block are removed before applying op^n. If all values are discard then the result is discard. If we do this for every block we are left with the set of all possible valuations defined over the variables $v_1 \cdots v_{n-1}$ each associated with a value (which was obtained by aggregating over the valuations of variable v_n in the block). We repeat the same procedure for variables v_{n-1} to v_1 to produce a final aggregate value. We define MAP$_B(I)$ to be this final aggregate value. The reader may also view this procedure as aggregation over variables in V by nesting aggregation operators from left (outermost) to right (innermost). i.e.

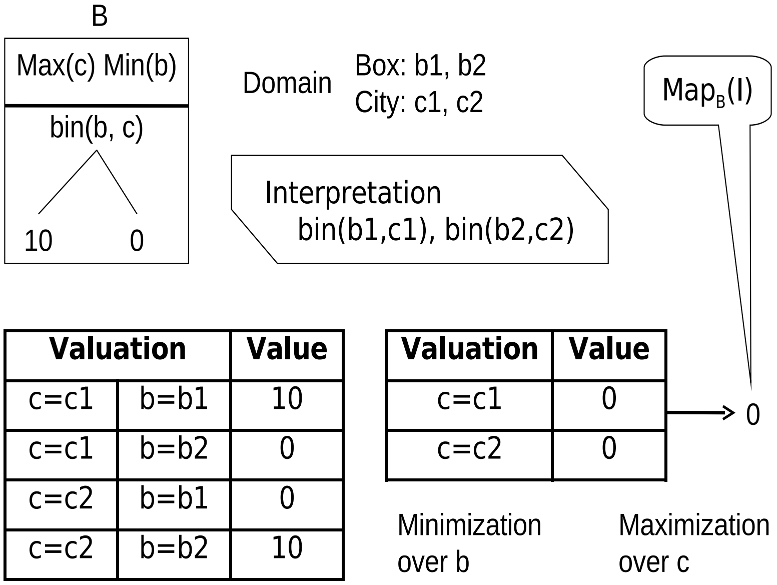
$$\text{MAP}_B(I) \quad = \quad op^1_{v_1}\left[op^2_{v_2}\left[\cdots\left[op^n_{v_n}\left[\text{MAP}_B\left(I,[v_1,v_2,\cdots v_n]\right)\right]\right]\cdots\right]\right]$$

The term in the center, MAP$_B(I, [v_1, v_2, \cdots v_n])$, is the value obtained by running a valuation defined by an assignment to the variables $v_1, \cdots v_n$ through B under I. In order to reduce the notational clutter, in the rest of the chapter we will drop brackets so that the above equation looks as follows

$$\begin{aligned} \text{MAP}_B(I) &= op^1_{v_1} op^2_{v_2} \cdots op^n_{v_n}[Map(I,[v_1,v_2,\cdots v_n])] \\ &= op^1_{v_1} op^2_{v_2} \cdots op^{n-1}_{v_{n-1}} op^n[c_1^{[v_1 \cdots v_{n-1}]} \cdots c_m^{[v_1 \cdots v_{n-1}]}] \end{aligned}$$

where each $c_i^{[v_1 \cdots v_{n-1}]}$ is a value corresponding to a different object assignment to variable v_n in the block defined by the values assigned to the variables $v_1 \cdots v_{n-1}$.

Figure 8.1 shows an example GFODD B capturing the following statement from the logistics domain: $\exists c \forall b$, box b is in city c. The output of B is 10 if all boxes are in one city and 0 otherwise. In the example GFODD shown, $V = \{Max(c), Min(b)\}$. Aggregation is done from right to left, one variable at a time. In the example, therefore, given the value



Valuation		Value
c=c1	b=b1	10
c=c1	b=b2	0
c=c2	b=b1	0
c=c2	b=b2	10

Valuation	Value
c=c1	0
c=c2	0

Figure 8.1: A Generalized FODD Example

of MAP$_B(I, \zeta)$ for every possible valuation ζ, MAP$_B(I)$ is calculated by first aggregating the values MAP$_B(I, \zeta)$ over all bindings (or assignments) of the variable b, using the min operation, producing exactly one value per binding of variable c, and then aggregating all of the produced values over all bindings of variable c using the max operation. In this example, to keep the GFODD diagram simple, we assume the variables are typed and use only valuations that conform to the types of the variables. Had we used all possible valuations over the set of objects $\{b_1, b_2, c_1, c_2\}$, the diagram would have been more complicated as it would have had to represent $\exists c, \forall b, city(c) \wedge [box(b) \rightarrow bin(b, c)]$.

With decision diagrams employing these semantics, there are a few important points to note.

- The order in V is important. Changing the order of the variables can change the aggregation function and hence can change the map.

- First order decision diagrams (FODDs) form a proper subclass of GFODDs where the aggregation operator associated with every variable is max. In this case, due to properties of the max aggregation, the order of variables in V is not important.

- GFODDs with $0/1$ leaves can be used to express closed, function free first order formulas by employing the min aggregation operator over universally quantified variables and the max aggregation operator over existentially quantified variables.

Finally we want the functions represented by GFODDs to be well defined on any interpretation. For this we define:

Definition 6 *A GFODD B is legal iff it obeys the GFODD syntax and $\forall$ interpretations I, $\exists\zeta$, MAP$_B(I, \zeta) \neq d$.*

8.1.2 Combining GFODDs

So far we have focused on the syntax and semantics of GFODDs and their ability to represent complex functions over relational structures. The utility of such a representation, though, is in performing operations over such functions, for example max (taking the maximum), $+$ (addition) and $\times$ (multiplication) as used in the SDP algorithm. We call these operators *combination operators* and provide an algorithm Ex-apply to implement them. Notice that combination operators are different from aggregation operators. The next definition provides the intended meaning of combination.

Definition 7 *GFODD B is a combination of GFODDs B_1 and B_2 under the binary combination operator op_c iff $\forall$ interpretations I, MAP$_B(I) =$ MAP$_{B_1}(I)$ op_c MAP$_{B_2}(I)$.*

Aggregation and combination operators can interact, complicating the result of the combination operation. The following definition provides the condition under which a simple operation can perform the combination.

Definition 8 *A combination operator op_c and an aggregation operator op^a are a safe pair iff for any set of non-negative values $x_1, x_2, \ldots, x_k$ and any non-negative constant b it holds that*

$$op^a(x_1, x_2, \ldots, x_k)\ op_c\ b = op^a(x_1\ op_c\ b, x_2\ op_c\ b, \ldots, x_k\ op_c\ b)\ .$$

The aggregation operator max and combination operator $+$ form a safe pair because for any set $S = \{c_1 \cdots c_m\}$ and constant b, $max\{c_1 \cdots c_m\} + b = max\{c_1 + b, \cdots c_m + b\}$.

op_c	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\otimes$	$\otimes$	$\otimes$	$\otimes$	max	max	max	max
op^a	max	min	sum	avg	max	min	sum	avg	max	min	sum	avg
safe/unsafe	safe	safe	unsafe	safe	safe	safe	safe	safe	safe	safe	unsafe	unsafe

Table 8.1: List of safe and unsafe pairs for operators.

For example $max\{1, 9, 3, 12\} + 5 = max\{6, 14, 8, 17\} = 17$. The aggregation operator $mean$ and the combination operator max do not form a safe pair. For example $max\{mean\{1, 5, 3\}, 4\} = 4$ but $mean\{4, 5, 4\} = 4.33$.

Table 8.1 summarizes the safe and unsafe pairs for operators that are of interest to us. We later use the fact that the max and min aggregation operators are safe with all the combination operators listed.

We next turn to the algorithm for combining diagrams. FODDs can be combined with the *Apply* operation described in chapter 3. *Apply* produces a combination of 2 FODDs under combination operator op_c by choosing the smaller root (according to the FODD predicate order) to be the root of the resultant FODD and then recursing on the sub-diagrams. When the computation reaches the leaves in both diagrams, the result is op_c applied to the two leaf values. For GFODDs, if either leaf value is d, we define $v\ op_c\ d = d$, for any v so the discard value is carried over to the resulting leaf. We next define the combination procedure for GFODDs and prove its correctness.

Definition 9 *If $B_1 = \langle V_1, D_1\rangle$ and $B_2 = \langle V_2, D_2\rangle$ where V_1 and V_2 do not have any variables in common, and op_c is any combination operator, then Ex-apply(B_1, B_2, op_c) = $B = \langle V, D\rangle$, where*

1. *V is the aggregation function obtained by appending V_2 to V_1.*

2. *$D =$ apply(D_1, D_2, op_c).*

The next lemma shows that the combination produced by Ex-apply is correct when one of the diagrams is a single leaf with value b.

Lemma 18 *If $B = \langle V, D \rangle$ is a GFODD, b is a non-negative constant, op_c is a combination operator, and if for every aggregation operator op^a in V, (op^a, op_c) is a safe pair, then, $\forall$ interpretations I, MAP$_B(I)$ op_c $b = op^1_{v_1} op^2_{v_2} \cdots op^n_{v_n}[MAP_B(I, [v_2 \cdots v_n])$ op_c $b]$*

Proof: The proof is by induction on n, the number of operators (and variables) in V. By the semantics of GFODDs,

$$\text{MAP}_B(I) \;\; op_c \;\; b \;\; = \;\; op^1_{v_1} \cdots op^n_{v_n}[\text{MAP}_B(I, [v_1 \cdots v_n])] \;\; op_c \;\; b$$

When $n = 1$, we have

$$\begin{aligned} \text{MAP}_B(I) \;\; op_c \;\; b \;\; &= \;\; op^1_{v_1}[\text{MAP}_B(I, [v_1])] \;\; op_c \;\; b \\ &= \;\; op^1_{v_1}[\text{MAP}_B(I, [v_1]) \;\; op_c \;\; b] \end{aligned}$$

because op^1 and op_c form a safe pair. Assume that the statement is true for all V of $n-1$ or fewer aggregation operators. Consider a V with n aggregation operators. We then have,

$$\begin{aligned} \text{MAP}_B(I) \;\; op_c \;\; b \;\; &= \;\; op^1_{v_1} \cdots op^n_{v_n}[\text{MAP}_B(I, [v_1 \cdots v_n])] \;\; op_c \;\; b \\ &= \;\; op^1_{v_1}[c_1^{[v_1]} \cdots c_m^{[v_1]}] \;\; op_c \;\; b \\ &= \;\; op^1_{v_1}[c_1^{[v_1]} \;\; op_c \;\; b \;\; \cdots \;\; c_m^{[v_1]} \;\; op_c \;\; b] \end{aligned}$$

because op^1 and op_c form a safe pair. Here each $c_i^{[v_1]} = op^2_{v_2} \cdots op^n_{v_n}[\text{MAP}_B(I, [v_2 \cdots v_n])]$ for the i^{th} value of the variable v_1. By the inductive hypothesis we know that

$$op^2_{v_2} \cdots op^n_{v_n}[\text{MAP}_B(I, [v_2 \cdots v_n])] \;\; op_c \;\; b \;\; = \;\; op^2_{v_2} \cdots op^n_{v_n}[\text{MAP}_B(I, [v_2 \cdots v_n]) \;\; op_c \;\; b]$$

Thus,

$$\text{MAP}_B(I) \;\; op_c \;\; b \;\; = \;\; op^1_{v_1} op^2_{v_2} \cdots op^n_{v_n}[\text{MAP}_B(I, [v_2 \cdots v_n]) op_c \;\; b]$$

■

We now use this lemma to prove the correctness of Ex-apply.

Theorem 10 *Given GFODDs $B_1 = \langle V_1, D_1\rangle$ and $B_2 = \langle V_2, D_2\rangle$ and a combination operator op_c, if for every aggregation operator $op^a \in V_1 \cup V_2$, op^a and op_c form a safe pair, then $B = \langle V, D\rangle =$ Ex-apply(B_1, B_2, op_c) is a combination of functions B_1 and B_2 under operator op_c.*

Proof: Let $op^{i,j}$ and $v_{i,j}$ denote the i^{th} operator and variable respectively in V_j. V is a juxtaposition of V_1 and V_2 by the definition of Ex-apply. Therefore by the definition of the GFODD semantics, for any interpretation I,

$$\text{MAP}_B(I) \;=\; op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} op^{1,2}_{v_{1,2}} \cdots op^{n,2}_{v_{n,2}} [\text{MAP}_B(I, [v_{1,1} \cdots v_{n,1} v_{1,2} \cdots v_{n,2}])]$$

Since $D = apply(D_1, D_2, op_c)$, by the correctness of apply we have that for all interpretations I and valuations ζ, $\text{MAP}_B(I, \zeta) = \text{MAP}_{B_1}(I, \zeta)\ op_c\ \text{MAP}_{B_2}(I, \zeta)$. Also, since the variables in V_1 and V_2 are disjoint, we can write any valuation ζ as $\zeta_1\zeta_2$ such that ζ_1 is the sub-valuation of ζ over the variables in V_1 and ζ_2 is the sub-valuation of ζ over the variables in V_2. Thus we can write

$$\begin{aligned}\text{MAP}_B(I) \;=\;& op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} op^{1,2}_{v_{1,2}} \cdots op^{n,2}_{v_{n,2}} [\text{MAP}_{B_1}(I, [v_{1,1} \cdots v_{n,1}])\ \ op_c \\ & \text{MAP}_{B_2}(I, [v_{1,2} \cdots v_{n,2}])]\end{aligned}$$

Now the important observation is that since $\text{MAP}_{B_1}(I, [v_{1,1} \cdots v_{n,1}])$ does not depend on the variables in V_2, when aggregating over the variables in V_2, $\text{MAP}_{B_1}(I, [v_{1,1} \cdots v_{n,1}])$ can be treated as a constant. Since op_c forms a safe pair with all aggregation operators of V_2, by Lemma 18,

$$\begin{aligned}\text{MAP}_B(I) \;=\;& op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} (\text{MAP}_{B_1}(I, [v_{1,1} \cdots v_{n,1}])\ \ op_c\ \ op^{1,2}_{v_{1,2}} \\ & \cdots op^{n,2}_{v_{n,2}} (\text{MAP}_{B_2}(I, [v_{1,2} \cdots v_{n,2}]))) \\ =\;& op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}} (\text{MAP}_{B_1}(I, [v_{1,1} \cdots v_{n,1}])\ \ op_c\ \ \text{MAP}_{B_2}(I))\end{aligned}$$

Similarly when aggregating over variables in V_1, $\text{MAP}_{B_2}(I)$ can be treated as a constant because it does not depend on the value of any of the variables in V_1. Since op_c forms a safe pair with all the aggregation operators in V_1, by Lemma 18,

$$\begin{aligned}\text{MAP}_B(I) &= op^{1,1}_{v_{1,1}} \cdots op^{n,1}_{v_{n,1}}(\text{MAP}_{B_1}(I,[v_{1,1}\cdots v_{n,1}]))\ \ op_c\ \ \text{MAP}_{B_2}(I)\\ &= \text{MAP}_{B_1}(I)\ \ op_c\ \ \text{MAP}_{B_2}(I)\end{aligned}$$

Thus by definition, $B =$ Ex-apply(B_1, B_2, op_c) is a combination of B_1 and B_2 under the combination operator op_c. ■

The following theorem strengthens this result showing that Ex-apply has some freedom in reordering the aggregation operators while maintaining correctness. In particular if V_1 and V_2 are in "block form" e.g. $\exists^*\forall^*$, we can reorder the result V' to keep the same block form. This property is useful for our solution of RMDPs.

Theorem 11 *Let $B_1 = \langle V_1, D_1\rangle$ and $B_2 = \langle V_2, D_2\rangle$ be GFODDs that do not share any variables and assume that op_c forms a safe pair with all operators in V_1 and V_2. Let $B = \langle V, D\rangle =$ Ex-apply(B_1, B_2, op_c). Let V' be any permutation of V so long as the relative order of operators in V_1 and V_2 remains unchanged, and $B' = \langle V', D\rangle$. Then for any interpretation I, MAP$_B(I) =$ MAP$_{B'}(I)$.*

Proof: Let $V_1 = F^1_1 F^1_2 \cdots F^1_k$ and $V_2 = F^2_1 F^2_2 \cdots F^2_k$ so that each F^i_j is a series of zero or more consecutive aggregation operators in V_i. Then $V' = F^1_1 F^2_1 F^1_2 F^2_2 \cdots F^1_k F^2_k$ represents a permutation of V such that the relative order of operators in V_1 and V_2 remains unchanged. By the semantics of GFODDs,

$$\text{MAP}_{B'}(I) = F^1_1 F^2_1 \cdots F^1_k F^2_k \text{MAP}_{B'}(I, [v_{1,1}\cdots v_{n,1} v_{1,2}\cdots v_{m,2}])$$

where $v_{i,j,}$ is a variable in B_j. This is because B and B' share the diagram D and differ only in the aggregation function. Therefore,

$$\begin{aligned}\text{MAP}_{B'}(I) &= F^1_1 F^2_1 \cdots F^1_k F^2_k \text{MAP}_{B}(I, [v_{1,1}\cdots v_{n,1} v_{1,2}\cdots v_{m,2}])\\ &= F^1_1 F^2_1 \cdots F^1_k F^2_k [\text{MAP}_{B_1}(I,[v_{1,1}\cdots v_{n,1}])\ \ op_c\ \ \text{MAP}_{B_2}(I,[v_{1,2}\cdots v_{m,2}])]\end{aligned}$$

by the correctness of apply procedure . Since B_1 and B_2 do not share any variables, and op_c forms a safe pair with all operators in V_1 and V_2, we have the following sequence of equations where in each step we use Lemma 18 and the fact that one of the arguments is a constant with respect to the corresponding block of aggregation operators.

$$\begin{aligned}
\text{MAP}_{B'}(I) &= F_1^1 F_1^2 \cdots F_k^1[\text{MAP}_{B_1}(I,[v_{1,1}\cdots v_{n,1}])\ \ op_c\ \ F_k^2\text{MAP}_{B_2}(I,[v_{1,2}\cdots v_{m,2}])] \\
&= F_1^1 F_1^2 \cdots F_{k-1}^2[F_k^1\text{MAP}_{B_1}(I,[v_{1,1}\cdots v_{n,1}])\ \ op_c\ \ F_k^2\text{MAP}_{B_2}(I,[v_{1,2}\cdots v_{m,2}])] \\
&= \cdots \\
&= F_1^1 \cdots F_k^1\text{MAP}_{B_1}(I,[v_{1,1}\cdots v_{n,1}])\ \ op_c\ \ F_1^2\cdots F_k^2\text{MAP}_{B_2}(I,[v_{1,2}\cdots v_{m,2}])
\end{aligned}$$

Finally by Theorem 10, the last term is equal to MAP$_B(I)$ implying that MAP$_{B'}(I) =$ MAP$_B(I)$. ■

8.2 Model Checking Reductions for GFODDs

The R12 procedure introduced in Section 6.1 can be extended to operate on GFODDs. In this section we present extensions of R12 for two forms of aggregation functions. The first is a set of diagrams using only min aggregation. The second is the set of diagrams with max^*min^* aggregation. In this case the aggregation function consists of a series of zero or more max operators followed by a series of zero or more min operators. For this case we introduce two variants, $R12_d$ and $R12_0$, with differing computational costs and quality of reduction. We discuss each of these in turn starting with the R12 procedure for the min operator.

8.2.1 R12 for min aggregation

The case of min aggregation is obtained as a dual of the max aggregation case. Also the notion of instrumental paths here is the dual of the notion of instrumental paths for the max aggregation. However, it is worthwhile considering it explicitly as a building block for the next construction.

Definition 10 *If B is a GFODD with only the min aggregation function, and P is the DPO for B, then a path $p_j \in P$ is instrumental iff*

1. *there is an interpretation I and valuation, ζ, such that $Path_B(I, \zeta) = p_j$, and*
2. *$\forall$ valuations η, if $Path_B(I, \eta) = p_k$, then $k \leq j$.*

The generalized aggregation function for the min aggregation operator is the same as the max operator except that the max is replaced by the min and no special treatment is given to paths reaching the 0 leaf. We thus have a min^3 generalized aggregation function. The reduction procedure is identical to the max except that min^3 is used instead of max^3 and that edges in E have the targets replaced by $discard$ instead of 0. This is not strictly necessary, as we can replace the target of the edges with a large value (or ∞). But it is useful for the preparation of the next construction. A trivial adaptation of the proofs in the Chapter 6 yields the corresponding properties for min aggregation.

Lemma 19 *If a path p_i in GFODD B is instrumental under DPO PL, then $\exists$ I_o such that $\{$leaf(p_i), p_i, $I_o\} \in S$.*

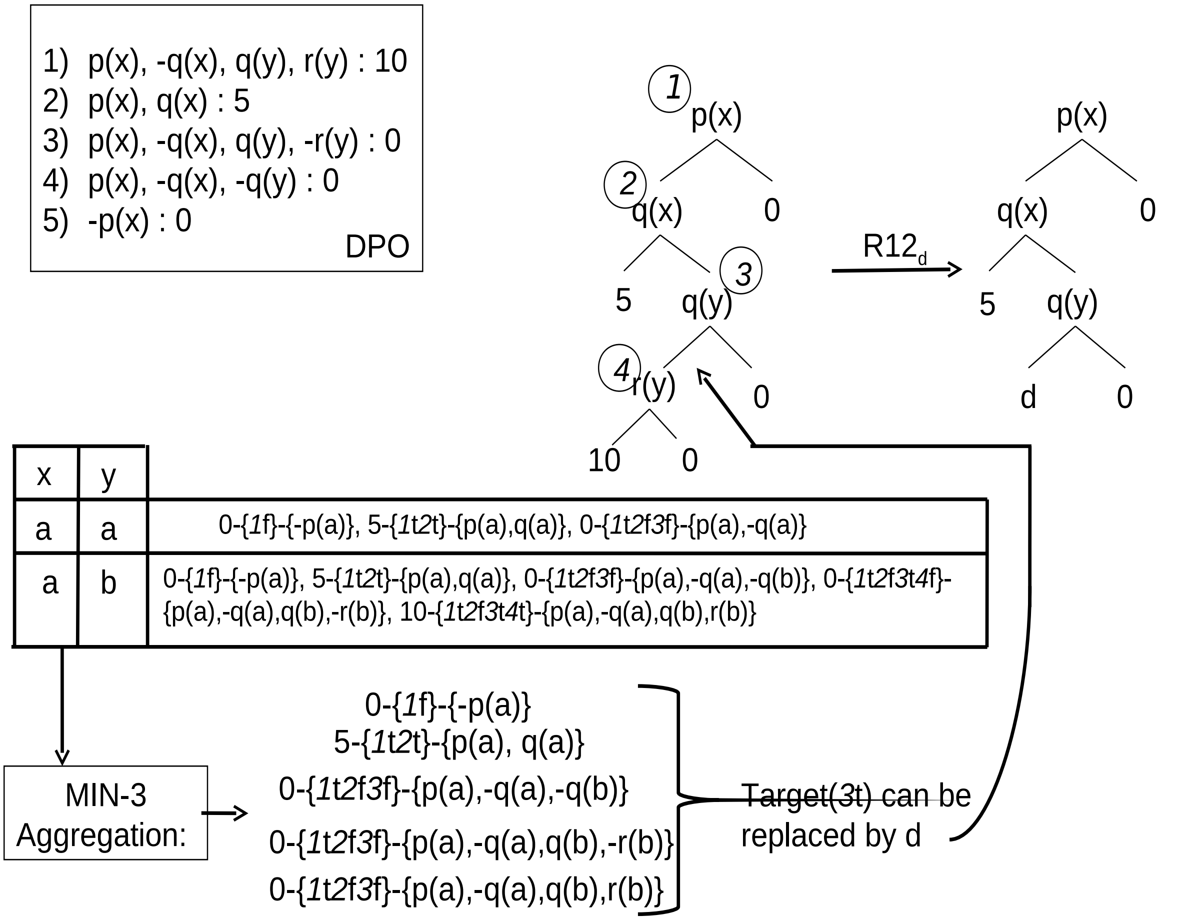
Lemma 20 *If there exists an instrumental path under PL that crosses e in B then $e \in E'$.*

Theorem 12 (soundness) *If GFODD B' is the output of R12(B) for any GFODD B, then $\forall$ interpretations I, MAP$_B(I)$ = MAP$_{B'}(I)$.*

Theorem 13 (completeness) *If no path crossing edge e and reaching a non-zero leaf in B is instrumental under PL, then R12 removes e.*

8.2.2 Model Checking Reduction for max^*min^* Aggregation

This section is concerned with GFODDs employing max^*min^* aggregation. The aggregation function consists of a series of zero or more max operators followed by a series of zero or more min operators. The aggregation function V is therefore split into V^l – the variables aggregated over using the max aggregation operator, and V^r – the variables aggregated over using the min aggregation operator. Thus, $V = V^lV^r$. The set U of all


1) p(x), -q(x), q(y), r(y) : 10
2) p(x), q(x) : 5
3) p(x), -q(x), q(y), -r(y) : 0
4) p(x), -q(x), -q(y) : 0
5) -p(x) : 0
DPO
1
p(x)
2
q(x)
0
5
3
q(y)
4
r(y)
0
10
0
R12d
p(x)
q(x)
0
5
q(y)
d
0
MIN-3 Aggregation:
0-{1f}-{-p(a)}
5-{1t2t}-{p(a), q(a)}
0-{1t2f3f}-{p(a),-q(a),-q(b)}
0-{1t2f3f}-{p(a),-q(a),q(b),-r(b)}
0-{1t2f3f}-{p(a),-q(a),q(b),r(b)}
Target(3t) can be replaced by d

x	y	
a	a	0-{1f}-{-p(a)}, 5-{1t2t}-{p(a),q(a)}, 0-{1t2f3f}-{p(a),-q(a)}
a	b	0-{1f}-{-p(a)}, 5-{1t2t}-{p(a),q(a)}, 0-{1t2f3f}-{p(a),-q(a),-q(b)}, 0-{1t2f3t4f}-{p(a),-q(a),q(b),-r(b)}, 10-{1t2f3t4t}-{p(a),-q(a),q(b),r(b)}

Figure 8.2: An example of reduction operator $R12_d$ for GFODDs with max^*min^* Aggregation. Each entry of the form value-$\{$path$\}$-$\{$interpretation$\}$ in the table expresses the value obtained by running the valuation of the corresponding row through the diagram under an equivalence class of interpretations. The min^3 aggregation function applied to every block (in this case there is just one block with $\zeta^l = a$ because there is only one variable x associated with the max aggregation operator) then calculates the possible aggregates that could be generated under different equivalence classes of interpretations. Since the edge *3t* does not appear along any of the instrumental paths leading to a non-zero leaf in the result of min^3, it is not instrumental and can be removed

possible valuations of the variables in B can be split into U^l and U^r, the sets of all valuations over the variables in V^l and V^r respectively. Any valuation $\zeta \in U$ can then be written as $\zeta^l\zeta^r$ where $\zeta^l \in U^l$ and $\zeta^r \in U^r$. Thus by the definition of GFODD semantics, for any interpretation I,

$$\begin{aligned} \text{MAP}_B(I) &= op^1_{v_1} \cdots op^n_{v_n}[\text{MAP}_B(I, [v_1 \cdots v_n])] \\ &= max_{\zeta^l \in U^l}[min_{\zeta^r \in U^r}[\text{MAP}_B(I, \zeta^l\zeta^r)]]. \end{aligned}$$

The procedure $R12_d$

Our first reduction operator captures a simple notion of instrumental paths. Considering the evaluation of B on I, we check whether a path is instrumental in obtaining the value of the min aggregation for any fixed ζ_l. If not then it clearly does not contribute to the final value. However, we must be careful when changing the value of the path, because this may affect min competitions (if we replace the value of the path with a value that is too low). We therefore use the value d in the reduction.

Definition 11 *If B is a GFODD with the max^*min^* aggregation function, and P is a DPO for B, then a path $p_i \in P$ is instrumental iff there is an interpretation I and valuation $\zeta = \zeta^l\zeta^r$, where $\zeta \in U$, $\zeta^l \in U^l$ and $\zeta^r \in U^r$, such that,*

1. *$Path_B(I, \zeta) = p_i$*
2. *For every $\eta^r \in U^r$, if $Path_B(I, \zeta^l\eta^r) = p_j$, then $j \leq i$ under P*

The $R12_d$ procedure for the max^*min^* aggregation is identical to the R12 procedure for the min aggregation with the following exceptions.

1. The set U of valuations is built in the following way. Let O^l be the set of $|V^l|$ newly invented objects. Let O^r be the set of $|V^r|$ newly invented objects. O^l and O^r are disjoint. Let U^l be the sets of all possible valuations of the variables in V^l over the objects in O^l and let U^r be the set of all possible valuation of the variables in V^r over the objects in $O^l \cup O^r$. The set U is then defined as $U = \{\zeta^l\zeta^r \mid \zeta^l \in U^l$ and $\zeta^r \in U^r\}$.

2. The set S is defined as $S = \bigcup_{\zeta^l}$Reduction-Aggregation(B, U_{ζ^l}, PL), where U_{ζ^l} is the block of valuations corresponding to ζ^l. Thus the set Val in the procedure is divided into blocks, each containing a set of valuations with the same ζ^l. S is the union of the sets generated as a result of applying min^3 to each blocks of Val.

Figure 8.2 shows a small example of this reduction. The process is similar to the R12 procedure for the max aggregation, except for the generalized aggregation function. A DPO is first established as shown. Sets $O^l = \{1\}$ and $O^r = \{2\}$ are invented and the table (Val) is generated by running the $getValue$ procedure on the valuations generated from those. Finally, since Val consists of a single block (since only one variable is associated with the max operator), $min^3(Val)$ is evaluated to produce the 5 $\{$leaf, path, Interpretation$\}$ triplets as shown. For example combining 0-$\{1t2f3f\}$-$\{p(a)$,$\neg q(a)\}$ with 10-$\{1t2f3t4t\}$-$\{p(a)$,$\neg q(a)$, $q(b)$,$r(b)\}$ under min^3 we get 0-$\{1t2f3f\}$-$\{p(a)$,$\neg q(a)$, $q(b)$,$r(b)\}$. The targets of all edges other than the ones present in the paths of the resultant triplets can be replaced by the value d. The resultant diagram is shown.

The proof of correctness follows the same outline as above but accounts for the extra aggregation operators.

Lemma 21 *If a path p_i in GFODD B employing the max^*min^* semantics is instrumental under DPO PL, then $\exists$ I_o such that $\{$leaf(p_i), p_i, $I_o\} \in S$.*

Proof: If p_i is instrumental under PL then $\exists$ interpretation I over a set of objects O_I and valuation $\zeta = \zeta^l\zeta^r$ such that $Path_B(I, \zeta) = p_i$ and for every η^r, if $Path_B(I, \zeta^l\eta^r) = p_j$, then $j \leq i$ under P. Let O'^l be the set of objects that participate in ζ^l and let O'^r be the set of objects that participate in ζ^r but not in ζ^l. Clearly $1 \leq |O'^l| \leq |V^l|$ and $1 \leq |O'^r| \leq |V^r|$. Let $o'^l_1 \in O'^l$ and $o'^r_1 \in O'^r$. Add $|V^l| - |O'^l|$ new objects to O'^l and $|V^r| - |O'^r|$ new objects to O'^r.

Construct interpretation I' by first projecting I to include only the objects in O'^l and O'^r and then defining truth values and predicates over the new objects in O'^l and O'^r to behave identical to o'^l_1 and o'^r_1 respectively. Let O'^l and O'^r be the sets O^l and O^r used in the $R12_d$ procedure to generate the set of valuations, U. U can be split into blocks so that each valuation $\eta = \eta^l\eta^r$ belonging to U can be assigned to the block corresponding to η^l. Let U_{ζ^l} be the block corresponding to ζ^l.

Since $\zeta \in U_{\zeta^l}$, and I' contains the relevant portion of I, ζ traverses p_i under I'. Additionally if $\exists \eta \in U_{\zeta^l}$ such that $Path_B(I', \eta) = p_j$, and $j > i$ under PL, we could construct another valuation $\hat{\eta} = \hat{\eta^l}\hat{\eta^r}$ by replacing the new objects in $\hat{\eta^l}$ and $\hat{\eta^r}$ by $o_1^{\prime l}$ and $o_1^{\prime r}$ respectively, so that $Path_B(I, \hat{\eta}) = p_j$. However, we know that no such $\hat{\eta}$ exists. Therefore there is no $\eta \in U_{\zeta^l}$ such that $Path_B(I', \eta) = p_j$, and $j > i$ under PL.

Let $I_o = \bigcup_{\eta \in U_{\zeta^l}} (PF(Path_B(I', \eta)))\eta$. That is, I_o includes all the atoms of I' that participate in traversing paths in B for all valuations in U_{ζ^l}. By construction the corresponding parts $(PF(Path_B(I', \eta)))\eta$ will be included in the $valueset$ returned by the getValue procedure. Clearly $I_o \subseteq I'$. Therefore if I' is consistent then so is I_o. If Val_{ζ^l} is the block in Val corresponding to the valuations in U_{ζ^l}, then by the definition of min^3, $min^3(Val_{\zeta^l})$ must contain an entry $\{$leaf(p_i), p_i, $I_o\}$. Finally since $min^3(Val_{\zeta^l})$ is a subset of S, S must contain $\{$leaf(p_i), p_i, $I_o\}$. ■

Lemma 22 *If there exists an instrumental path under PL that crosses e in B then $e \in E'$.*

Proof: If there is an instrumental path $p_i \in PL$ that crosses edge e, by Lemma 21 $\exists$ I_o such that $\{$leaf(p_i), p_i, $I_o\} \in S$. By definition of E', $e \in E'$. ■

Theorem 14 (soundness) *If GFODD B' is the output of $R12_d$(B) for any GFODD B with the max^*min^* semantics, then $\forall$ interpretations I, MAP$_B(I)$ = MAP$_{B'}(I)$.*

Proof: By the definition of $R12_d$, the only difference between B and B' is that some edges that pointed to sub-FODDs in B, point to the $discard$ leaf in B'. These are the edges in the set E at the end of the $R12_d$ procedure. Therefore any valuation crossing these edges is discarded from the aggregation function. Valuations not crossing these edges will achieve the same value in B' as they did in B.

Fix any interpretation I over any set O_I of objects. Let U be the set of all valuations of the variables in B over O_I. Each valuation $\eta \in U$ can be expressed as $\eta = \eta^l\eta^r$ such that $\eta^l \in U^l$ and $\eta^r \in U^r$. MAP$_B(I)$ can then be expressed as

$$\text{MAP}_B(I) \quad = \quad max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_B(I, \eta^l\eta^r)]$$

Now for any $\eta^l \in U^l$, let p_i be a path such that $\exists \eta^r \in U^r$, $Path_B(I, \eta^l\eta^r) = p_i$ and $\forall \iota^r \in U^r$, $Path_B(I, \eta^l\iota^r) = p_j$ implies that $j \leq i$ under the same DPO employed in the

$R12_d$ reduction procedure. By definition p_i is instrumental and hence by Lemma 22 none of the edges on p_i are affected by $R12_d$. Therefore MAP$_B(I, \eta^l \eta^r) =$ MAP$_{B'}(I, \eta^l \eta^r)$ and by the property of min aggregation, for η^l,

$$min_{\eta^r \in U^r}[\text{MAP}_B(I, \eta^l \eta^r)] \;\; = \;\; min_{\eta^r \in U^r}[\text{MAP}_{B'}(I, \eta^l \eta^r)].$$

Since this is true for every $\eta^l \in U^l$, it is also true for the aggregation, that is

$$max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_B(I, \eta^l \eta^r)] \;\; = \;\; max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_{B'}(I, \eta^l \eta^r)].$$

Therefore MAP$_B(I) =$ MAP$_{B'}(I)$. ■

The Procedure $R12_0$

The introduction of the discard value in the leaves makes handling and interpretation of diagrams awkward. In this section we show that at some additional computational cost this can be avoided. With some extra bookkeeping, a variant of the R12 procedure can avoid replacing edge targets with the d leaf and in the process, potentially remove more redundancies from a $max^* min^*$ GFODD. To motivate the new procedure, consider what happens during evaluation of interpretation I on GFODD B. Each block b of valuations corresponding to a ζ^l is collapsed under min aggregation. Let P_b (P_{ζ^l}) denote the set of paths in B traversed by the valuations in b and ordered by the given DPO. We can view this procedure as a competition among the paths in P_b. The winner of this competition is the path of highest index in P_b. Denote this path by p_b (p_{ζ^l}). The min competition applied to all blocks creates a "super block" $\hat{b}$ of all the winners, each corresponding to a ζ^l. Finally all the ζ^ls are collapsed under the max aggregation. This process can, in turn, be viewed as a max competition among the paths in $P_{\hat{b}}$. The winner of this competition is the path with the least index in $P_{\hat{b}}$. Obviously this path also wins the min competition in its own block. We call this block the max block b^* and the winning path p_{b^*}. Then, MAP$_B(I) =$ leaf(p_{b^*}). We observe the following:

1. If the value of the leaf reached by any path in the max block is *reduced* to a value at least as large as leaf(p_{b^*}), the map remains unchanged. This is because the min

competition on the max block will still produce the same result. Additionally, since we are only reducing the values of other paths, none of the other blocks will produce a winner with a leaf value higher than leaf(p_{b^*}).

2. If the value of the leaf reached by any path in any block b other than the max block is reduced to 0, leaf(p_{b^*}) will still win the max competition and the map will be preserved.

The above observation suggests that we can reduce a GFODD in the following way,

1. Preserve the targets of all edges in all paths winning the final max competition under any interpretation. We call these *instrumental* edges.

2. Identify edges on paths in B that appear in the max block under any possible interpretation I. We call these ***block*** edges. For each block edge e, replace target(e) by a value that is (1) at least as large as leaf(p_{b^*}) under I but (2) no larger than the smallest leaf reachable by traversing e. Notice that (1) means that p_{b^*} wins the min competition of the blocks and (2) makes sure we never add value to any path. The condition (2) is somewhat limiting in that it may prevent us from reducing the diagram because of conflict with (1). One might be able to prune further using a deeper analysis; however (2) provides a cheap test that appears to provide reasonable coverage.

3. Replace the targets of all other edges by 0.

In the remainder of this section, we describe the $R12_0$ reduction procedure and prove its correctness. The input to the procedure is a GFODD $B = \langle V, D \rangle$ and a DPO PL for B. The output is a reduced GFODD B'. We redefine the generalized aggregation functions min^3 and max^3 to capture the bookkeeping needed for block edges. min^3 is redefined as follows.

min^3: min^3 takes as input a set Val of valuations each corresponding to a set of $\langle value$, $path$, $interpretation\rangle$ triplets. The output is a set of all possible quadruplets $\langle v_o$, p_o, E_o, $I_o\rangle$ generated as follows

1. Let $X = \{\langle v_1, p_1, I_1\rangle, \cdots, \langle v_{|Val|}, p_{|Val|}, I_{|Val|}\rangle\}$ be a set constructed by picking one triplet from the set corresponding to each valuation $\zeta \in Val$.

2. $v_o = min[v_1, \cdots, v_{|Val|}]$

3. p_o is the path of least index under DPO PL that appears in a triplet in X such that leaf(p_o) $= v_o$.

4. E_o is the set of all the edges appearing in all the paths in all the triplets in X except the edges in p_o

5. $I_o = \bigcup_i I_i$ such that I_o is consistent and $\langle v_i, p_i, E_i, I_i\rangle \in X$.

max^3**:** max^3 takes as input a set Val of valuations each corresponding to a set of $\langle value, path, EdgeList, interpretation\rangle$ quadruplets. The output is a set of all possible quadruplets $\langle v_o, p_o, E_o, I_o\rangle$ generated as follows.

1. Let $X = \{\langle v_1, p_1, E_1, I_1\rangle, \cdots, \langle v_{|Val|}, p_{|Val|}, E_{|Val|}, I_{|Val|}\rangle\}$ be a set constructed by picking one quadruplet from the set corresponding to each valuation $\zeta \in Val$.

2. $v_o = max[v_1, \cdots, v_{|Val|}]$

3. p_o is the path of highest index under DPO PL that appears in a quadruplet in X such that leaf(p_o) $= v_o$.

4. E_o is the set E_i such that $p_o = p_i$ and $v_o = v_i$

5. $I_o = \bigcup_i I_i$ such that I_o is consistent and $\langle v_i, p_i, E_i, I_i\rangle \in X$.

Note that the set E_o in min^3 collects the edges from the losing paths in one block whereas the set E_o in max^3 collects the block edges.

The $R12_0$ procedure is as follows.

1. The set U of valuations is built in the following way. Let O^l be the set of $|V^l|$ newly invented objects. Let O^r be the set of $(|V^l|^{|V^l|} + 1)|V^r|$ newly invented objects. O^l and O^r are disjoint. Let U^l be the sets of all possible valuations of the variables in V^l over the objects in O^l and let U^r be the set of all possible valuation of the variables

in V^r over the objects in $O^l \cup O^r$. The set U is then defined as $U = \{\zeta^l\zeta^r \mid \zeta^l \in U^l$ and $\zeta^r \in U^r\}$.

2. For every edge we maintain 3 variables. low(e) and high(e) are bounds on its value and InstrEdge(e) is a flag. These are initialized as follows. For all edges e in B, set low(e) $= -1$, high(e) $= l_e$, where l_e is the value of the smallest leaf reachable through e in B, and InstrEdge(e) $= 0$.

3. Run the $maxmin^3$ procedure as follows.

   (a) Divide Val into $|U^l|$ blocks of valuations each block corresponding to a valuation $\zeta^l \in U^l$. Let X be the set of these blocks.

   (b) Let $Y = \{\langle \zeta^l - min^3(b) \rangle \mid \zeta^l \in U^l$ and $b \in X$ is the block corresponding to $\zeta^l\}$.

   (c) Let $S = max^3(Y)$.

   (d) For every quadruplet $\langle v_o, p_o, E_o, I_o \rangle \in S$, do

      i. For every edge $e \in p_o$, set InstrEdge(e) $= 1$.

      ii. For every edge $e \in E_o$, set low(e) to max[low(e), v_o].

4. Finally the target of every edge e is replaced as follows:

   (a) If InstrEdge(e) $= 1$, do not replace.

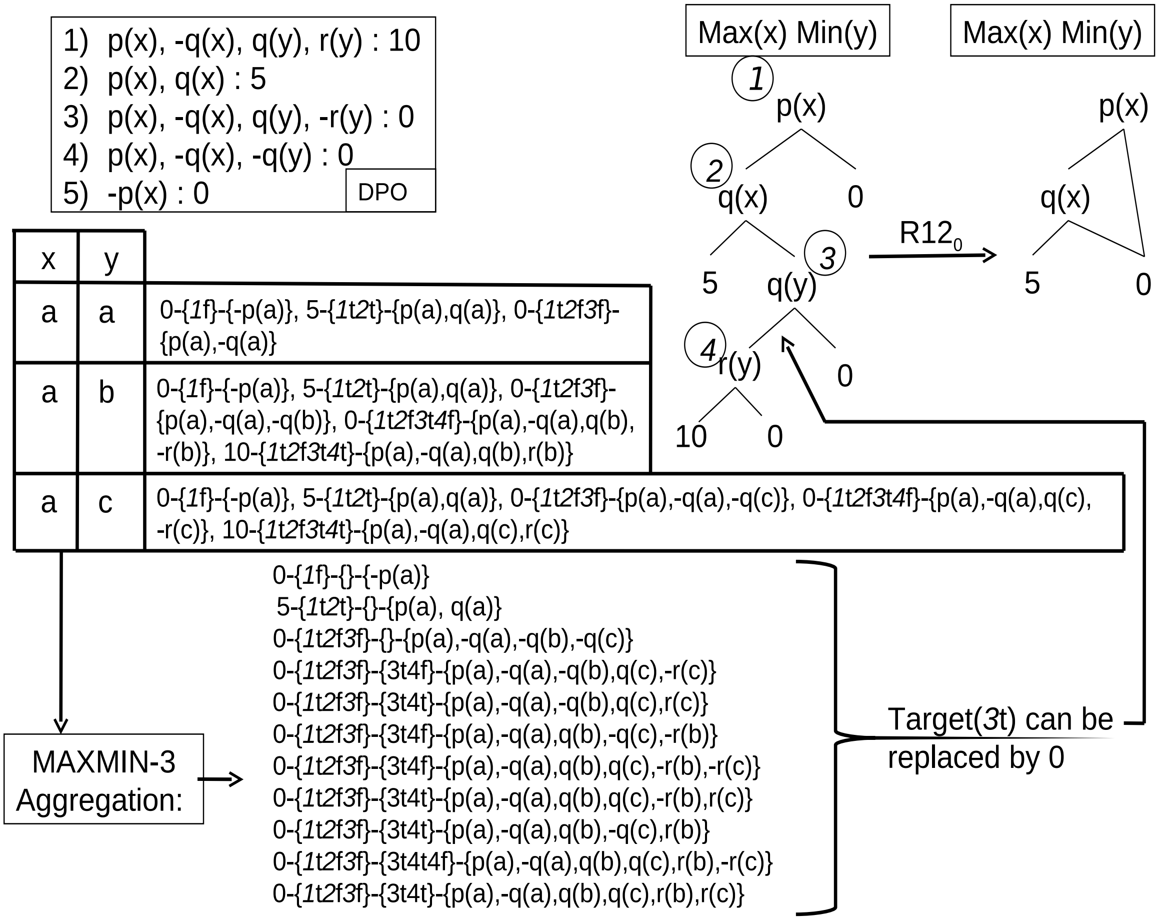
   (b) If InstrEdge(e) $= 0$, low(e) $\neq -1$ i.e. e is a block edge and high(e) $\geq$ low(e), replace target(e) by any suitable value v, such that low(e) $\leq v \leq$ high(e).

   (c) If InstrEdge(e) $= 0$ and low(e) $= -1$, i.e. e is not a block edge replace target(e) by 0.

Figure 8.3 shows an example of the $R12_0$ reduction.

In the remaining part of this section we provide a proof of soundness for $R12_0$. To that end we introduce the following terms.

Definition 12 *An edge e in GFODD B is instrumental iff $e \in p_{b^*}$ under some interpretation.*



1) p(x), -q(x), q(y), r(y) : 10
2) p(x), q(x) : 5
3) p(x), -q(x), q(y), -r(y) : 0
4) p(x), -q(x), -q(y) : 0
5) -p(x) : 0

x	y	
a	a	0-{*1*f}-{-p(a)}, 5-{*1*t*2*t}-{p(a),q(a)}, 0-{*1*t*2*f*3*f}-{p(a),-q(a)}
a	b	0-{*1*f}-{-p(a)}, 5-{*1*t*2*t}-{p(a),q(a)}, 0-{*1*t*2*f*3*f}-{p(a),-q(a),-q(b)}, 0-{*1*t*2*f*3*t*4*f}-{p(a),-q(a),q(b),-r(b)}, 10-{*1*t*2*f*3*t*4*t}-{p(a),-q(a),q(b),r(b)}
a	c	0-{*1*f}-{-p(a)}, 5-{*1*t*2*t}-{p(a),q(a)}, 0-{*1*t*2*f*3*f}-{p(a),-q(a),-q(c)}, 0-{*1*t*2*f*3*t*4*f}-{p(a),-q(a),q(c),-r(c)}, 10-{*1*t*2*f*3*t*4*t}-{p(a),-q(a),q(c),r(c)}

0-{*1*f}-{}-{-p(a)}
5-{*1*t*2*t}-{}-{p(a), q(a)}
0-{*1*t*2*f*3*f}-{}-{p(a),-q(a),-q(b),-q(c)}
0-{*1*t*2*f*3*f}-{3t4f}-{p(a),-q(a),-q(b),q(c),-r(c)}
0-{*1*t*2*f*3*f}-{3t4t}-{p(a),-q(a),-q(b),q(c),r(c)}
0-{*1*t*2*f*3*f}-{3t4f}-{p(a),-q(a),q(b),-q(c),-r(b)}
0-{*1*t*2*f*3*f}-{3t4f}-{p(a),-q(a),q(b),q(c),-r(b),-r(c)}
0-{*1*t*2*f*3*f}-{3t4t}-{p(a),-q(a),q(b),q(c),-r(b),r(c)}
0-{*1*t*2*f*3*f}-{3t4t}-{p(a),-q(a),q(b),-q(c),r(b)}
0-{*1*t*2*f*3*f}-{3t4t4f}-{p(a),-q(a),q(b),q(c),r(b),-r(c)}
0-{*1*t*2*f*3*f}-{3t4t}-{p(a),-q(a),q(b),q(c),r(b),r(c)}

Figure 8.3: An example of reduction operator $R12_0$ for GFODDs with max^*min^* Aggregation. The initial diagram is the same as in Figure 8.2. This time $|O^r| = 3$ because $|V^l| = |V^r| = 1$ and hence $|V^l| + (|V^l|^{|V^l|} + 1)|V^r| = 3$. Each entry of the form value-{path}-{interpretation} in the table expresses the value obtained by running the valuation of the corresponding row through the diagram under an equivalence class of interpretations. The MAXMIN-3 aggregation function then calculates the possible aggregates that could be generated under different equivalence classes of interpretations. Since we have only one block, we only need to run the extended min^3 aggregation on this example. The result is shown below the table. For example the entries 0-$\{1t2f3f\}$-$\{p(a), \neg q(a)\}$, 10-$\{1t2f3t4t\}$-$\{p(a), \neg q(a), q(b), r(b)\}$ and 10-$\{1t2f3t4t\}$-$\{p(a), \neg q(a), q(c), r(c)\}$, give the last row in the result. The edges *3t*,*4t* and *4f* are identified as a block edges. For edge *3t*, InstrEdge(*3t*) $= 0$ because there is no winner of the max block contains edge *3t*. high(*3t*) $= 0$ because the smallest leaf reachable by traversing *3t* is 0. The $maxmin^3$ procedure sets low(*3t*) to 0 because the highest leaf reached by any path defeating the paths containing *3t* in the max block is 0. Thus target(*3t*) can be set to 0 without violating the constraint low(*3t*) $\leq$ target(*3t*) $\leq$ high(*3t*). Setting the target of *3t* to 0 reduces the diagram. Note that in this example all edges shown are block edges because there is only one block - the max block. All the edges appearing in the result of $maxmin^3$ are instrumental edges and their targets are preserved by the reduction procedure

Definition 13 *An edge e in GFODD B is a block edge if it is not instrumental and $e \in path \in P_{b^*}$ under some interpretation.*

Definition 14 *An edge e in GFODD B is a useless edge if it is neither an instrumental edge nor a block edge.*

Definition 15 *For any block edge e, CannotExceed(e) is the value of the smallest leaf reachable through e and CannotLag(e) is the value of the largest leaf(p_{b^*}) over all interpretations, when a path containing e appears in the max block*

Definition 16 *A reduction procedure R that reduces a given GFODD B to produce GFODD B' is block-safe if it conforms to the following rules.*

1. *R identifies all instrumental edges in B and for each such identified edge e, R maintains target(e).*

2. *R identifies all block edges in B and for each such identified edge e, R replaces target(e) by any leaf value v such that CannotLag(e) $\leq v \leq$ CannotExceed(e).*

3. *For each edge e that is not identified by R as an instrumental or block edge, R replaces target(e) by 0.*

The proof below has two parts. Theorem 15 says that any reduction procedure that is *block-safe* must also be a sound reduction procedure. Theorem 16 proves that $R12_0$ is *block safe*.

Theorem 15 *If reduction procedure R is block-safe and $B' = R(B)$, then for every interpretation I, MAP$_B(I) =$ MAP$_{B'}(I)$.*

Proof: Fix any interpretation I over any set O_I of objects. Let U be the set of all valuations of the variables in B over O_I. Let $\zeta = \zeta^l\zeta^r \in U$ be a valuation traversing p_{b^*} in B. MAP$_B(I)$ can then be expressed as

$$\begin{aligned}
\text{MAP}_B(I) &= max_{\eta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_B(I, \eta^l\eta^r)]] \\
&= max[min_{\zeta^r \in U^r}[\text{MAP}_B(I, \zeta^l\zeta^r)], max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_B(I, \eta^l\eta^r)]]]
\end{aligned}$$

Since the definition of *block-safe* guarantees that the target of every edge e is not replaced by a value greater than CannotExcede(e), target(e) only decrease in value. Therefore, for any valuation $\eta \in U$, leaf($Path_B(I,\eta)$) $\geq$ leaf($Path_{B'}(I,\eta)$). Therefore we have,

$$max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_{B'}(I,\eta^l\eta^r)]] \leq max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_{B}(I,\eta^l\eta^r)]]$$

Additionally, the definition of *block-safe* guarantees that all instrumental edges are preserved and that the value reached by the block edges is never reduced below leaf(p_{b^*}). Therefore, ζ reaches leaf(p_{b^*}) in both B and B'. No other valuation in the max block $b*$ reaches a value less than leaf(p_{b^*}) when evaluated on B'. Thus,

$$\begin{aligned} min_{\zeta^r \in U^r}[\text{MAP}_{B'}(I,\zeta^l\zeta^r)] &= min_{\zeta^r \in U^r}[\text{MAP}_{B}(I,\zeta^l\zeta^r)] \\ &= leaf(p_{b^*}) \end{aligned}$$

Finally,

$$\begin{aligned} \text{MAP}_{B'}(I) &= max[min_{\zeta^r \in U^r}[\text{MAP}_{B'}(I,\zeta^l\zeta^r)], max_{\eta^l \neq \zeta^l \in U^l}[min_{\eta^r \in U^r}[\text{MAP}_{B'}(I,\eta^l\eta^r)]]] \\ &= min_{\zeta^r \in U^r}[\text{MAP}_{B'}(I,\zeta^l\zeta^r)] \\ &= leaf(p_{b^*}) \\ &= \text{MAP}_{B}(I) \end{aligned}$$

■

It is clear that $R12_0$ identifies some instrumental edges and some block edges. To show that this is true for all such edges over an infinite set of interpretations some of which have infinite domains, we show that each such edge is discovered by one of the finite combinations in our procedure. Thus even if two edges are the block edges of the same p_{b^*}, they may be discovered by different I_o's in our procedure. This is achieved in the following theorem by showing that any instrumental edge (on p_{b^*}) and block edge (on p_j) are appropriately accounted for by $R12_0$.

Theorem 16 (soundness) *$R12_0$ is block-safe.*

Proof: Line 4 in the $R12_0$ procedure enumerates the treatment of different edges in B. It remains to be shown that

1. If an edge e in B is instrumental under some interpretation I, then $R12_0$ sets Instr(e) $= 1$.

2. If an edge e is a block edge under some interpretation I, then $R12_0$ sets low(e) $\geq$ CannotLag(e).

3. If an edge e is a block edge under some interpretation I, then $R12_0$ sets high(e) $\leq$ CannotExceed(e).

Of the above, 3 is true by the definition of $R12_0$. Consider any interpretation I. Let $\zeta = \zeta^l\zeta^r$ be a valuation traversing $p_{b*} = p_i$ in B under I. Let $\eta = \zeta^l\eta^r$ be any other valuation in the max block $b*$ that does not win the min competition and let $Path_B(I, \eta) = p_j$. Let O'^l be the set of objects that participate in ζ^l and define the set $O'^r = \{\ o \notin O'^l \mid o$ participates in η^r or in ι^r, where ι^l contains only the objects from O'^l and $\iota^l\iota^r$ wins the min competition in its block$\}$. By construction $|O'^l| \leq |V^l|$ and $|O'^r| \leq (|V^l|^{|V^l|} + 1)|V^r|$. Let $o_1'^l \in O'^l$ and $o_1'^r \in O'^r$. Add $|V^l| - |O'^l|$ new objects to O'^l and $(|V^l|^{|V^l|} + 1)|V^r| - |O'^r|$ new objects to O'^r.

Construct interpretation I' by first projecting I to include only the objects in O'^l and O'^r and then defining truth values and predicates over the new objects added to O'^l and O'^r to behave identical to $o_1'^l$ and $o_1'^r$ respectively. Let O'^l and O'^r be the sets O^l and O^r used in the $R12_0$ procedure to generate the set of valuations, U.

Since I' contains the relevant portion of I, $Path_B(I', \zeta) = p_i$ and $Path_B(I', \eta) = p_j$. Additionally if there exists valuation $\zeta^l\iota^r \in U$ such that $Path_B(I', \zeta^l\iota^r) = p_k$ and $k > i$ under PL, then we could construct another valuation $\hat{\zeta}^l\hat{\iota}^r$ by replacing the new objects in ζ^l and ι^r by $o_1'^l$ and $o_1'^r$ respectively so that $Path_B(I, \hat{\zeta}^l\hat{\iota}^r) = p_k$. However, we know that there is no such $\hat{\zeta}^l\hat{\iota}^r$. In other words, path p_i is the winner of the min competition in the max block $b*$ under I'. An identical argument proves that if b is a block in U corresponding to ι^l, then p_b defined relative to I is the winner of the min competition in b under I'.

Let $I_{\iota^l\iota^r} = PF(Path_B(I', \iota^l\iota^r))\iota^l\iota^r$ be the set of atoms on the path $p_{\iota^l\iota^r}$ in B traversed by some valuation $\iota^l\iota^r$ under I'. By construction, a triplet $\langle$leaf($p_{\iota^l\iota^r}$),$p_{\iota^l\iota^r}$,$I_{\iota^l\iota^r}\rangle$

appears in the output of the getValue procedure, when run on $\iota^l\iota^r$. Therefore, by the definition of min^3, the set X generated by applying min^3 to b must contain an entry $\langle$leaf(p_b),p_b,E_b,$I_b\rangle$, where $I_b = \bigcup_{\iota^r \in U^r} (PF(Path_B(I', \iota^l\iota^r)))\iota^l\iota^r$. Similarly the set produced by applying min^3 to the max block must contain an entry $\langle$leaf(p_{b^*}),p_{b^*},E_{b^*},$I_{b^*}\rangle$, where $I_{b^*} = \bigcup_{\iota^r \in U^r} (PF(Path_B(I', \zeta^l\iota^r)))\zeta^l\iota^r$. Also, E_{b^*} must contain all the edges in p_j.

Now, by the definition of max^3, the set S built in the reduction procedure must contain an entry $\langle$leaf(p_{b^*}),p_{b^*},E_{b^*},$I_o\rangle$ where $I_o = \bigcup_{\iota \in U} (PF(Path_B(I', \iota)))\iota$ is consistent because it is a subset of I'.

Therefore $e \in p_{b^*}$ is marked instrumental by $R12_0$. Every edge $e \in p_j$ is marked with low(e) $\geq$ leaf(p_{b^*}). Since the choice of I, p_{b^*} and p_j was arbitrary in the above argument, this holds for all block edges, implying that low(e) $\geq$ CannotLag(e). Thus $R12_0$ is *block-safe*. ■

We have proved above that $R12_d$ and $R12_0$ are both sound reductions, the question of completeness of these reductions is still open. Since theorem proving of first order formulas with the quantifier settings $\exists^*\forall^*$ is decidable, there is hope for a completeness result.

8.3 An Application of GFODDs for Value Iteration in Relational MDPs

So far we have described a general theory of GFODDs. This included the syntax and semantics of GFODDs, combination procedures and reduction procedures for GFODDs. In this section we describe an application of GFODDs for solving RMDPs.

The algorithm is very similar to the one presented in Chapter 3 but uses the more expressive GFODDs instead of FODDs as the underlying representation for the RMDP. Recall that domain dynamics are expressed as Truth Value Diagrams (TVD) which describe, for each deterministic alternative of each probabilistic action and for each world predicate, the conditions under which the corresponding world literal becomes `true` when the action is executed and that action alternative occurs. Figure 8.4 shows an example of a TVD for the parametrized world predicate $p(A, B)$ under the deterministic action $A(x^*, y^*)$ in a hypothetical planning domain that we use to illustrate the algorithm.

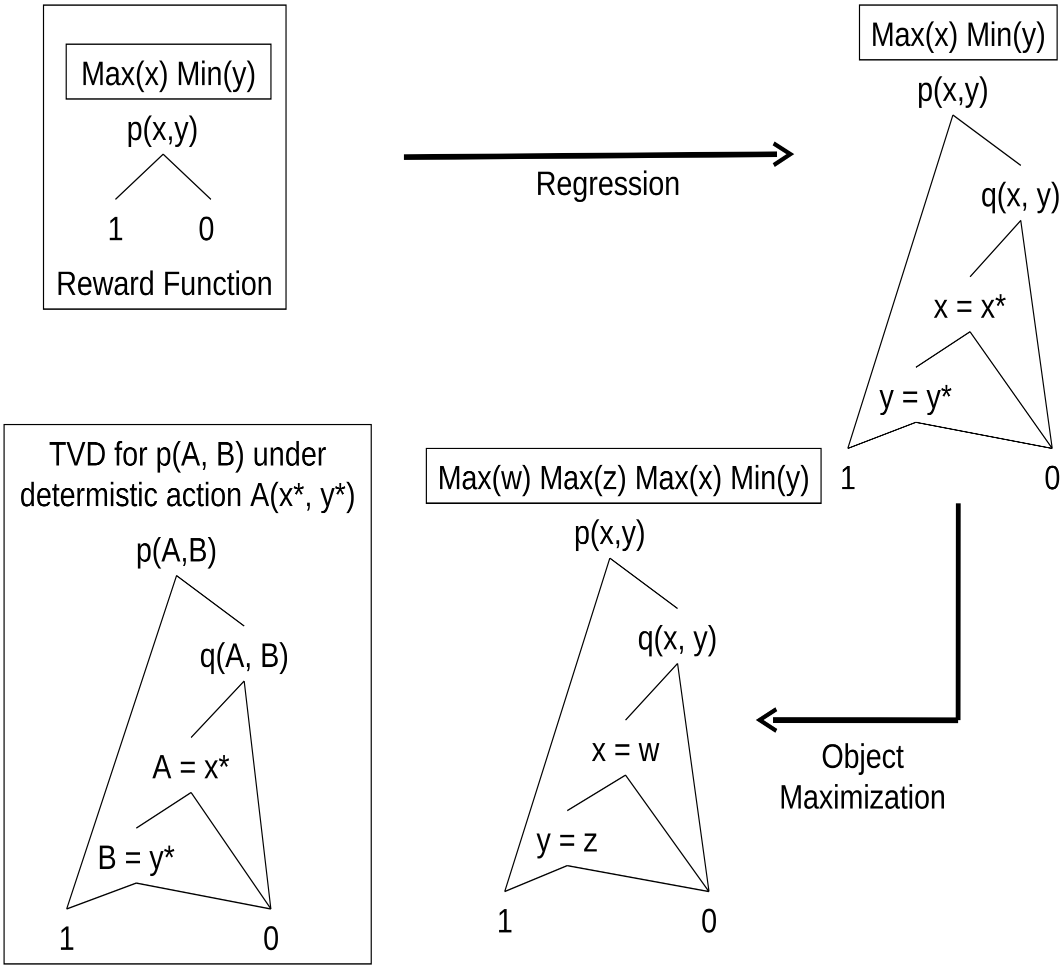


Figure 8.4: Example of GFODD Regression and Object Maximization

8.3.1 The VI-GFODD Algorithm

In this section we show that the FODD based VI algorithm can be generalized to handle cases where the reward function is described by a GFODD with $max^{*}min^{*}$ aggregation. The following are the steps of the algorithm *VI-GFODD*. A subsequent discussion shows why *VI-GFODD* produces the correct result at each step.

1. **Regression:** The n step-to-go value function V_n is regressed over every deterministic variant $A_j(\vec{x})$ of every action $A(\vec{x})$ to produce $Regr(V_n, A(\vec{x}))$ by replacing each node in V^{n-1} by its corresponding Truth Value Diagram (TVD) without changing the aggregation function. This step is unchanged.

2. **Add Action Variants:** The Q-function $Q_{V_n}^{A(\vec{x})} = R \oplus [\gamma \otimes \oplus_j (prob(A_j(\vec{x})) \otimes Regr(V_n, A_j(\vec{x})))]$ for each action $A(\vec{x})$ is generated by combining regressed diagrams using Ex-apply. This step is changed simply by using Ex-apply instead of Apply.

3. **Object Maximization:** Maximize over the action parameters of $Q_{V_n}^{A(\vec{x})}$ to produce $Q_{V_n}^{A}$ for each action $A(\vec{x})$, thus obtaining the value achievable by the best ground instantiation of $A(\vec{x})$. This step is implemented by converting action parameters in $Q_{V_n}^{A(\vec{x})}$ to variables each associated with the max aggregation operator, and prepending these operators to the aggregation function.

4. **Maximize over Actions:** The $n+1$ step-to-go value function $V_{n+1} = \max_A Q_{V_n}^{A}$, is generated by combining the diagrams using Ex-apply.

Figure 8.4 shows an example of the VI algorithm using GFODDs for a simplified domain. This domain contains one single deterministic action. Therefore steps 2 and 4 of the algorithm are not needed. The reward function is regressed over the deterministic action $A(x^*, y^*)$, which is defined such that $p(x, y)$ is true after the action if it was true before or if $q(x, y)$ was true before and the action performed was $A(x, y)$. The reward is 1 if $\exists x$, $\forall y$, $p(x, y)$ and 0 otherwise. Since the action can make at most one $p(x, y)$ true at a time, intuitively, the regressed diagram should capture the union of the following conditions for returning a value of 1.

1. $\exists x$, $\forall y$, $p(x, y)$

2. $\exists x$, such that for all but one y, $p(x, y)$ is true and for that y, $q(x, y)$ is true.

Figure 8.4 shows the diagram after being regressed and object maximized. The final diagram is correct because it returns a 1 iff one of above situations occur. If $\exists x$, $\forall y$, $p(x, y)$, then all valuations in the blocks with that value of x and fixed values for w and z will reach the 1 leaf directly from the root. Evaluating $Min(y)$ will collapse these blocks to partial valuations with a 1 value. Now since the rest of the aggregation is maximization, the 1 value will be returned as the map. If $\exists x$, such that for all but one y, $p(x, y)$ is true and for that y, $q(x, y)$ is true, then all valuations in the blocks with that value of x, the other values of y and fixed values for w and z reach a 1 leaf directly through the root. The valuation in the block with the one value of y would traverse right from the root but would still reach the 1 leaf depending on the condition $w = x$ and $z = y$. Note that there will be exactly one block where this valuation will reach the 1 leaf. Evaluating $Min(y)$ would collapse that block into a valuation with value 1. Since the rest of the aggregation is maximization, the 1 value will be returned as the map. When neither of the conditions is true, there will be at least one valuation in every block that reaches a 0 leaf. Hence evaluating $Min(y)$ would collapse every block to a valuation with a 0 value.

For Value Iteration to work correctly with GFODDs, all the steps of the algorithm listed above must be correct. Regression by block replacement is correct regardless of the aggregation function. Recall that a TVD for a predicate under deterministic action $A_j(\vec{x})$ describes conditions under which the predicate becomes `true` after $A_j(\vec{x})$ is executed. In chapter 3 we imposed the constraint that TVDs cannot include free variables. Using this constraint the diagrams before and after regression have exactly the same variables. Also in chapter 3 we showed that regression is correct for any valuation.

Lemma 23 *(Wang, 2008) If V_n is the n step to go value function, BR-regress$(V_n, A(\vec{x}))$ is the result of regressing V_n over the deterministic action $A(\vec{x})$, and ζ is any valuation to the variables of V_n (and thus also the variables of BR-regress$(V_n, A(\vec{x}))$), then $MAP_{V_n}(s, \zeta)$ $= MAP_{\textit{BR-regress}(V_n,A(\vec{x}))}(\hat{s}, \zeta)$*

The lemma claims that after regression the map of every valuation stays the same for any interpretation. Therefore, under any interpretation, the map of V_n before and after regression is also the same irrespective of the aggregation function on V_n.

The third step of Object Maximization is correct because converting action parameters in $Q_{V_n}^{A(\vec{x})}$ to variables each associated with the max aggregation operator, and prepending these operators to the aggregation function of $Q_{V_n}^{A}$, implies that the map of $Q_{V_n}^{A}$ under any interpretation will now be the map of $Q_{V_n}^{A(\vec{x})}$ maximized over all possible values of the action parameters, as required. Steps 2 and 4 are correct by the correctness of Ex-apply. Since value iteration requires combining diagrams under the $\oplus$, $\otimes$ and the max operators, only GFODDs with aggregation operators that are safe with the combination operators $\oplus$, $\otimes$ and max may be used. Thus aggregation operators max and min can be used. The current version of VI-GFODD does not allow aggregation operators like $\sum$, $mean$ etc.

Thus we have a correct Value Iteration algorithm for GFODDs with max and min aggregations. Finally, Theorem 11 guarantees that if we start with a reward function GFODD with an aggregation of the form $max^{*}min^{*}$, then throughout Value Iteration all GFODDs produced to have an aggregation function of the same form. With the R12 reductions for this case, we have a sound procedure that can help keep the diagrams compact over the Value Iteration process. We have therefore shown:

Theorem 17 *For any Relational MDP where the aggregation function of the reward function diagram contains only operators that are safe with the combination operators $+$, $\times$ and max, algorithm VI-GFODD produces the correct value function at every iteration.*

Corollary 1 *For any Relational MDP where the reward function has a $max^{*}min^{*}$ aggregation, VI-GFODD produces the correct value function at every iteration and the R12 procedure can be used to reduce diagrams such that the final result also has $max^{*}min^{*}$ aggregation.*

8.4 Summary and Concluding Remarks

This chapter significantly extends the representation power of FODDs. We show how Generalized FODDs allow for arbitrary aggregation functions, thereby facilitating representation of complex functions, and how basic operations on them can be performed. In particular we can naturally capture and manipulate logical formulas with existential and universal quantifiers using $\max$ and $\min$ aggregation. In addition we show that first order Value Iteration can be supported in a more expressive setting when the MDP is represented by GFODDs. This new formulation can naturally handle universal goals that were handled heuristically by previous implementations of first order Value Iteration (Sanner & Boutilier, 2009) and by the FODD-PLANNER.

As we have shown, the idea of model-checking reductions extends to model-checking reduction operators for a useful subset of GFODDs. Nevertheless, more work is needed to identify efficient reductions for this subset and for other interesting subsets of GFODDs.

Chapter 9

Conclusions

The work in this thesis has focused on compact representations for sequential decision making problems. In particular we derived motivation from problem domains like logistics where the state space is large, action effects are probabilistic and objectives can be complex but there exists rich relational structure that can be leveraged by solution algorithms. Relational Markov decision processes (RMDP) have become a popular tool for solving such problems. To this end we follow the approach of Boutilier et al. (2001), who developed the Symbolic Dynamic Programming (SDP) algorithm to solve RMDPs. The main idea in SDP is to abstract the relational structure of the underlying domain and generate a solution in terms of the relational structure rather than actual domain objects. Such a solution is independent of the actual problem instance and is valid for all domain sizes. The RMDP solution algorithm we presented in this thesis is an instance of SDP and shares all the advantages of SDP that arise out of abstraction. Our main contribution is the development of compact and expressive knowledge representations to capture the reward structure and domain dynamics of the underlying RMDP and appropriate algorithms for efficient implementation of SDP using this representation.

Through theoretical results and emperical evidence, we have shown our approach to be practical and useful. There are many open questions that naturally arise from this work. We discuss some of the interesting questions in Section 9.2. First the following section presents a summary of the contributions of this thesis.

9.1 Summary of Contributions

This thesis makes the following contributions.

1. **First Order Decision Diagrams:** In Chapter 3 we invented First Order Decision Diagrams (FODD) by modifying and extending the approach of Groote and Tveretina (2003). FODDs are a compact knowledge representation for real valued functions over relational structures and are useful in defining utilities and probabilities of world states in an RMDP. We presented the FODD representation and algorithms to manipulate them. We demonstrated the use of FODDs to represent RMDPs and developed an SDP algorithm for the FODD representation. However, FODDs are a very general representation and we believe they can be employed in a variety of applications that involve structured representations.

2. **Theorem Proving Reductions:** Our RMDP solver based on FODDs performs reasoning by combination of FODDs. Reasoning with First Order models is an integral part of all SDP based algorithms (Boutilier et al., 2001; Kersting et al., 2004; Sanner & Boutilier, 2009). This reasoning process creates diagrams that are large and contain a lot of redundant structure. Indeed any application that requires performing reasoning with FODDs will face this problem. Our contribution to mitigate this issue and improve the practical applicability of FODDs is the development of logical simplification operators for FODDs in Chapter 4. These “reduction” operators use First Order logical implication as the primary tool to identify and remove redundant structure from the diagram. Therefore we call them Theorem Proving Reductions.

3. **Model Checking Reductions:** Proving First Order logical implication is an expensive operation. Moreover any constraints in the domain must be explicitly specified to the implication engine otherwise it can fail to prove true statements. These issues can limit the applicability of FODDs. To mitigate these issues, in Chapter 6, we introduced a new paradigm for reduction of FODDs based on model-checking and proved its superiority over theorem proving reductions. We presented theoretical and practical versions of model-checking reductions and provided proofs of correctness and completeness. The practical versions of model-checking reductions are very

efficient and, we believe, have utility in a variety of applications using structured representations.

4. **Weighted Goal Ordering Heuristic:** To apply SDP to concrete planning problems, one typically plans for abstract generic goals and then uses goal decomposition to put the abstract solutions together for a concrete instance. In Chapter 5, we introduced the Weighted Goal Ordering heuristic and showed its superiority over the heuristic of Sanner and Boutilier (2009) in domains where goal serializability is a crucial factor.

5. **FODD-Planner:** In Chapter 5, using theorem proving reductions to simplify FODDs, we presented a prolog based software system that implements the SDP algorithm with the FODD representation. We demonstrated this system by solving stochastic planning problems showing performance comparable to top ranking systems from the International Planning Competitions.

6. **Self-Taught Planning:** In Chapter 7 we developed a new paradigm for planning by learning. The idea is to provide FODD-PLANNER with a small "training set" of world states of interest, but no indication of optimal actions in any states. The FODD-PLANNER uses this training set and performs logical simplification by model-checking reductions. We also showed that such training examples can be constructed on the fly from a description of the planning problem. Thus we bootstrap our planner to get a self-taught planning system. By combining this idea with model-checking reductions we showed drastic improvements in planning efficiency over the use of theorem proving reductions on a variety of IPC domains. Although we employed the FODD-PLANNER to demonstrate the self-taught planning paradigm we believe that this technique is applicable with any SDP based system.

7. **Generalized FODDs:** FODDs are compact and expressive but when considered as logical formulas they are limited to existential quantification. To address this we introduced Generalized FODDs (GFODD) in Chapter 8, where we extend the representation power to arbitrary aggregation or quantification. GFODDs are very expressive structures that share the same compactness advantages as FODDs. We discussed several properties of GFODDs and presented reductions for an important subset of

GFODDs. We also identified conditions under which GFODDs can be composed or combined by a simple procedure. Similar to FODDs, we believe GFODDs can be employed in a variety of applications that involve structured representations.

8. **VI-GFODD:** We have shown that the same SDP based algorithm that employs FODDs as the underlying representation is also valid when GFODDs are used as the underlying representation. We proved the correctness of this algorithm, VI-GFODD, for a very expressive subset of GFODDs. Within this representation we can capture objectives like *transport at least one box to Paris and all trucks to London* in the logistics domain. This was not possible with FODDs.

9.2 Future Directions

Although we have covered many issues in this thesis, there are many questions that arise from this work. Broadly we can classify these into two categories covering short term and long term research questions.

Under the short term category, an important issue is that of building efficient evaluation rountines for policies represented as FODDs. Similar to other SDP based algorithms (Sanner & Boutilier, 2009), FODD-PLANNER generates an abstract policy for a given planning domain. This policy uses a goal decomposition heuristic to aggregate information about the utility of the current world state w.r.t. various decomposed parts of the goal. The process of evaluating the individual utility values and combining them has been the bottleneck in the plan execution routine of FODD-PLANNER and improving the efficiency of this task will be instrumental in building a better evaluation routine. Two possible directions to resolve this are employing faster subsumption engines and caching the utility values of recently evaluated world states.

Another issue in this category is that of ordering atoms in a FODD. Similar to ADDs, the order of atoms in a FODD can have a dramatic effect on its size. Discovering the optimal order for a given domain is a hard problem (it is NP-Hard for propositional ADDs).

In this thesis we have put the burden of inventing the optimal order on the user of FODD-PLANNER. However, one might hope for an inexpensive heuristic solution, following common practice in propositional decision diagrams (Rudell, 1993).

Two more issues in this category arise from the self-taught planning paradigm. One is the employment of SDP based RMDP solvers other than FODD-PLANNER, in self-taught planning. The other and more interesting question is that of alternative methods of example generation. One idea is to generate the set of training examples from solutions of multiple planners (teachers), each specializing in a different part of the state space. It would be interesting to see how such a method would enhance the performance of the planning system because in the self-taught planning formalism, contradictory information from different teachers does not adversely affect the learned model. Ideas from the research on active learning might also prove useful in acquiring training examples. A related problem for future study is that of characterizing the quality of the training set. Intuitively the best training set is the one in which all important conditions are identified by the fewest examples. It would also be interesting to see if PAC learning bounds on sample complexity can be derived for practical model-checking reductions.

The work on GFODDs also leads to two questions in the short term research category corresponding to two current limitations of GFODDs - composing functions defined by GFODDs in the general case and performing logical simplification of GFODDs efficiently. We believe that techniques for composition can be developed for $mean$ and $\sum$ aggregation. This will greatly widen the applicability of GFODDs. Efficient logical simplification will make GFODDs a very powerful, expressive, compact and efficiently manipulable tool for applications with structured representations. Deriving intuitions and ideas from the work on logical simplification of FODDs based on model-checking could be useful here.

The category of long term research questions contains many interesting problems. Most of these arise from our work on GFODDs, a largely unexplored area. The most obvious among them is the use of expressive GFODDs to represent and solve RMDPs. Answering this question, however, is not straight forward given the inherent limitation of representing solutions in factored MDPs (Allender et al., 2002) and RMDPs. Therefore using forward search methods similar to UCT (Kocsis & Szepesvari, 2006) in conjunction with dynamic programming might prove useful. We have already seen success in blending search and

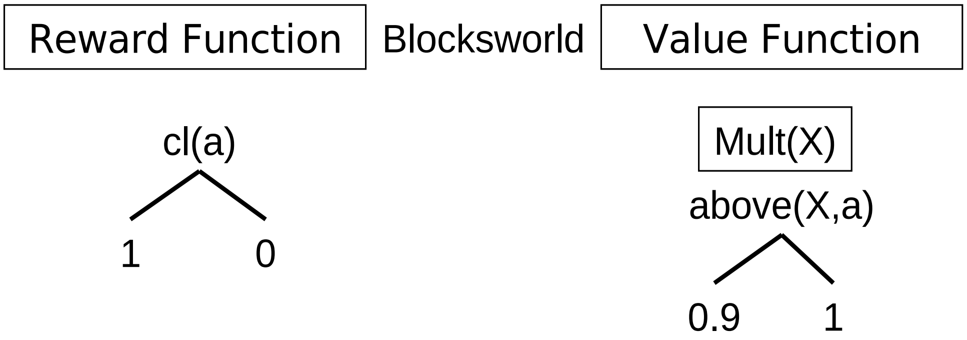


Figure 9.1: Reward and Value function for the goal $cl(a)$ in the blocksworld domain

dynamic programming with FODD-PLANNER in the self-taught planning paradigm and we believe there is potential in this line of research towards developing practical, compact, expressive and efficient RMDP solvers. Additionally, in context of the connection between SDP and EBL as explained in Chapter 2, GFODDs might also prove useful in addressing issues related to the "Generalization to N problem". For instance, Kersting et al. (2004) showed an example in the blocksworld domain where the goal is to make a particular block, a, clear ($cl(a)$) and the value function is infinite in size because there could be any number of blocks on top of a. However, the value function can be represented compactly using GFODDs in conjunction with a more descriptive predicate, $above$, as shown in Figure 9.1. In the figure, $above(X, a)$ is true for any block X that is part of a tower stacked on top of block a, aggregation over X is performed by the simple multiplication operator and the discount factor is 0.9. Thus the multiplicative aggregation implicitly captures the number of steps to the goal. Although the existence of a compact value function does not imply an efficient algorithm to produce it, at least in this particular case we know that the problem is not inherently that of representation.

Another issue in the long term research category is that of solving games. The current work on RMDPs is limited to a single agent and applicable only when the environment is non-adversarial. However, many real world domains are best described by the multi-agent adversarial setting e.g. real time strategy games, network security and military planning. Here it is important to consider the actions of the adversaries and generate a policy to achieve the agent's goals or thwart the opponent's goals. The seminal works in this area (Dietterich & Flann, 1997; Schaeffer, Burch, Bjornsson, Kishimoto, Muller, Lake, Lu, &

Sutphen, 2007) are based on the complementary approaches of SDP and search. Yet the representations they use are much simpler in comparison with what RMDP solvers are capable of. Similar to the problems solved by RMDPs, however, the agent objectives in games can be complex (e.g. for all friend soldiers, maximize the total number of enemy soldiers captured by each while minimizing physical harm to oneself). This suggests an advantage in using the GFODDs to address such problems. To our knowledge, there is no method that takes advantage of relational structure of games, generalizes over problem size and yet achieves complex objectives. In a similar vein to the approaches mentioned above, employing the RMDP solution with GFODDs in conjunction with forward search to solve games holds promise. GFODDs naturally handle min and max which are the main combining functions needed to solve games.

Another research area where FODDs and GFODDs can be applied is that of Statistical Relational Models (SRM) (Getoor & Tasker, 2007). SRMs for probabilistic reasoning have recently become popular due to their succinct representation, easier learning due to shared parameters and lifted inference methods. In fact, the relational VI algorithm of Boutilier et al. (2001) and the implementation of this algorithm using (G)FODDs can be seen to perform some form of lifted inference in probabilistic models. Recently several algorithms that take advantage of model structure in inference have been proposed (Poole, 2003; Braz, Amir, & Roth, 2005; Jaimovich, Meshi, & Friedman, 2007; Milch, Zettlemoyer, Kersting, Haimes, & Kaelbling, 2008; Singla & Domingos, 2008; Sen, Deshpande, & Getoor, 2008, 2009; Kersting, Ahmadi, & Natarajan, 2009; Kisynski & Poole, 2009). The advantage of lifted inference techniques is that they use abstraction to identify repeated factors in computation and thus lead to speedup. Comparatively, research in dynamic models is at a fairly exploratory stage (Kersting, DeRaedt, & Raiko, 2006) and the techniques of lifted inference have not been applied to different classes of problem such as reinforcement learning. These models, in various form, need to represent real valued functions over relational structures. Hence employing GFODDs to represent these functions thus creating a scope for complex yet compact and efficiently manipulable functions could be beneficial. As described in this thesis, GFODDs have the ability to perform reasoning with dynamic models. Thus GFODDs could be the missing link between SRMs and dynamic SRMs. There can be a number of ways to leverage GFODDs in this area. One idea is to represent the entire SRM as a set of GFODDs. Lifted inference can then be performed naturally by the manipulation of GFODDs.

We note, however, that in order to develop GFODD based solutions to these research problems we need to first solve the problem of composing functions defined by GFODDs with expressive aggregation functions and performing logical simplification of GFODDs efficiently.

Bibliography

Allender, E., Kearns, M., Arora, S., Russell, A., & Moore, C. (2002). A note on the representational incompatibility of function approximation and factored dynamics. In *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 431–437.

Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., & Somenzi, F. (1993). Algebraic decision diagrams and their applications. In *IEEE /ACM ICCAD*, pp. 188–191.

Balla, R., & Fern, A. (2009). UCT for tactical assault planning in real-time strategy games. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 40–45.

Barto, A., Bradtke, S., & Singh, S. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1-2), 81–138.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Blum, A., & Furst, M. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90(1-2)*, 279–298.

Blum, A., & Langford, J. (1998). Probabilistic planning in the Graphplan framework. In *Proceedings of the European Conference on Planning*, pp. 8–12.

Bonet, B., & Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, *129*(1-2), 5–33.

Boutilier, C., Dean, T., & Hanks, S. (1996). Planning under uncertainty: Structural assumptions and computational leverage. In *Proceedings of the European Workshop on Planning*, pp. 157–171.

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

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1104–1111.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1999). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, *121*(1-2), 49–107.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in bayesian networks. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 115–123.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for First-Order MDPs. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 690–700.

Braz, R., Amir, E., & Roth, D. (2005). Lifted First-Order probabilistic inference. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1319–1325.

Bryant, R. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *C-35*(8), 677–691.

Bryant, R. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, *24*(3), 293–318.

Bryce, D., & Buffet, O. (2008) In *Online Proceedings of the Probabilistic track of IPC-06, http://ippc-2008.loria.fr/wiki/index.php/Main_Page*.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms*. MIT Press.

Croonenborghs, T., Ramon, J., Blockeel, H., & Bruynooghe, M. (2007). Online learning and exploiting relational models in reinforcement learning. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 726–731.

Dean, T., & Kanazawa, K. (1990). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*(3), 142–150.

Dearden, R. (2001). Structured prioritized sweeping. In *Proceedings of the International Conference on Machine Learning*, pp. 82–89.

DeJong, G., & Mooney, R. (1986). Explanation-Based Learning: An Alternative View. *Machine Learning*, *1*(2), 145–176.

Dietterich, T., & Flann, N. (1997). Explanation-based learning and reinforcement learning: A unified view. *Machine Learning*, *28*(2-3), 169–210.

Driessens, K., & Dzeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, *57*(3), 271–304.

Dzeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*(1-2), 7–52.

Feng, Z., & Hansen, E. (2002). Symbolic heuristic search for factored Markov decision processes. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 455–460.

Fern, A., Yoon, S., & Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, *25*(1), 75–118.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*(3-4), 189–208.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, *20*(1), 61–124.

Gardiol, N., & Kaelbling, L. (2003). Envelope-based planning in relational MDPs. In *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 1040–1046.

Garriga, G., Khardon, R., & De Raedt, L. (2007). On mining closed sets in multi-relational data. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 804–809.

Gelly, S., & Silver, D. (2007). Combining online and offline knowledge in UCT. In *Proceedings of the International Conference on Machine Learning*, pp. 273–280.

Gelly, S., & Wang, Y. (2006). Exploration exploitation in Go: UCT for monte-carlo Go. In *NIPS Workshop for On-line trading of Exploration and Exploitation*.

Gerevini, A., Bonet, B., & Givan, R. (2006) In *Online Proceedings of the Probabilistic track of IPC-05, http://www.ldc.usb.ve/ bonet/ipc5/docs/ipc-2006-booklet.pdf.gz*.

Getoor, L., & Tasker, B. (2007). *An Introduction to Statistical Relational Learning*. MIT Press.

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL: The planning domain definition language. Tech. rep., Yale Center for Computational Vision and Control.

Gretton, C., & Thiebaux, S. (2004). Exploiting First-Order regression in inductive policy selection. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 217–225.

Groote, J., & Tveretina, O. (2003). Binary decision diagrams for First-Order predicate logic. *Journal of Logic and Algebraic Programming*, *57*, 1–22.

Großmann, A., Hölldobler, S., & Skvortsova, O. (2002). Symbolic dynamic programming within the fluent calculus. In *Proceedings of the IASTED International conference on Artificial and Computational Intelligence*, pp. 378–383.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003a). Generalizing plans to new environments in relational MDPs. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1003–1010.

Guestrin, C., Koller, D., Parr, R., & Venkataraman, S. (2003b). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, *19*(1), 399–468.

Hanks, S., & McDermott, D. (1993). Modeling a dynamic and uncertain world i: Symbolic and probabilistic reasoning about change. *Artificial Intelligence*, *66*(1), 1–55.

Hansen, E., & Zilberstein, S. (2002). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*(1-2), 35–62.

Hoey, J., St-Aubin, R., Hu, A., & Boutilier, C. (1999). SPUDD: Stochastic planning using decision diagrams. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 279–288.

Hölldobler, S., Karabaev, E., & Skvortsova, O. (2006). FluCaP: a heuristic search planner for First-Order MDPs. *Journal of Artificial Intelligence Research*, *27*(1), 419–439.

Hölldobler, S., & Skvortsova, O. (2004). A logic-based approach to dynamic programming. In *Proceedings of the AAAI-04 workshop on learning and planning in Markov Processes – advances and challenges*.

Howard, R. (1960). *Dynamic Programming and Markov Processes.* MIT Press.

Jaimovich, A., Meshi, O., & Friedman, N. (2007). Template-based inference in symmetric relational Markov random fields. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 191–199.

Joshi, S., Kersting, K., & Khardon, R. (2009). Generalized First-Order decision diagrams for First-Order Markov decision processes. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1916–1921.

Joshi, S., Kersting, K., & Khardon, R. (2010). Self-taught decision theoretic planning with First-Order decision diagrams. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 89–96.

Joshi, S., & Khardon, R. (2008). Stochastic planning with First-Order decision diagrams. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 156–163.

Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1194–1201.

Kersting, K., Ahmadi, B., & Natarajan, S. (2009). Counting belief propagation. In *Proceedings of Uncertainty in Artificial Intelligence*.

Kersting, K., & De Raedt, L. (2004). Logical Markov decision programs and the convergence of logical TD(λ). In *Proceedings of Inductive Logic Programming*, pp. 180–197.

Kersting, K., DeRaedt, L., & Raiko, T. (2006). Logical hidden Markov models. *Journal of Artificial Intelligence Research*, *25*(1), 425–456.

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. In *Proceedings of the International Conference on Machine Learning*, pp. 465–472.

Kisynski, J., & Poole, D. (2009). Lifted aggregation in directed First-Order probabilistic models. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1922–1929.

Kocsis, L., & Szepesvari, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning*, pp. 282–293.

Koller, D., & Parr, R. (1999). Computing factored value functions for policies in structured MDPs. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1332–1339.

Koller, D., & Parr, R. (2000). Policy iteration for factored MDPs. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 326–334.

Krof, R. (1990). Real-time heuristic search. *Artificial Intelligence*, *42*(2-3), 189–211.

Kushmerick, N., Hanks, S., & Weld, D. (1995). An algorithm for probabilistic planning. *Artificial Intelligence*, *76*(1-2), 239–286.

Laird, J., Rosenbloom, P., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*(1), 11–46.

Little, I., & Thibaux, S. (2007). Probabilistic planning vs. replanning. In *Proceedings of the ICAPS Workshop on IPC: Past, Present and Future*.

Littman, M. (1997). Probabilistic propositional planning: Representations and complexity. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 748–754.

Littman, M., & Younas, H. (2004) In *Online Proceedings of the Probabilistic track of IPC-04, http://www.cs.rutgers.edu/mlittman/topics/ipc04-pt/proceedings/.*

Lloyd, J. (1987). *Foundations of Logic Programming*. Springer Verlag. Second Edition.

Majercik, S., & Littman, M. (2003). Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, *147*(1-2), 119–162.

Maloberti, J., & Sebag, M. (2004). Fast theta-subsumption with constraint satisfaction algorithms. *Machine Learning*, *55*(2), 137–174.

Martelli, A., & Montanari, U. (1973). Additive and/or graphs. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1–11.

Mausam, & Weld, D. (2003). Solving relational MDPs with First-Order machine learning. In *Proceedings of the ICAPS Workshop on Planning under Uncertainty and Incomplete Information*.

McDermott, D. (1998) In *Online Proceedings of AIPS planning competition, ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html*.

McMillan, K. L. (1993). *Symbolic model checking*. Kluwer Academic Publishers.

Milch, B., Zettlemoyer, L., Kersting, K., Haimes, M., & Kaelbling, L. (2008). Lifted probabilistic inference with counting formulas. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1062–1068.

Minton, S. (1988). *Learning Search Control Knowledge: An explanation-based approach*. Kluwer Academic Publishers.

Mitchell, T., Keller, T., & Kedar-Cabelli, S. (1986). Explanation-Based Generalization: A Unifying View. *Machine Learning*, *1*(1), 47–80.

Moore, A., & Atkeson, C. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, *13*(1), 103–130.

Morales, E., & Sammut, C. (2004). Learning to fly by combining reinforcement learning with behavioral cloning. In *Proceedings of the International Conference on Machine Learning*, p. 76.

Nilsson, N. (1971). *Problem-solving methods in artificial intelligence*. McGraw-Hill.

Pednault, E. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the Conference on Knowledge Representation and Reasoning*, pp. 324–332.

Penberthy, J., & Weld, D. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Conference on Knowledge Representation and Reasoning*, pp. 103–114.

Poole, D. (2003). First-Order probabilistic inference. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 985–991.

Price, B., & Boutilier, C. (2003). Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, *19*, 569–629.

Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. Wiley.

Puterman, M., & Shin, M. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, *24*, 1127–1137.

Rivest, R. (1987). Learning decision lists. *Machine Learning*, *2*, 229–246.

Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the International Conference on Computer-Aided Design*, pp. 42–47.

Russel, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach. Third Edition*. Prentice Hall Series in Artificial Intelligence.

Sanner, S. (2008). *First-Order decision-theoretic planning in structured relational environments*. Ph.D. thesis, University of Toronto.

Sanner, S., & Boutilier, C. (2006). Practical linear value-approximation techniques for First-Order MDPs. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 409–417.

Sanner, S., & Boutilier, C. (2009). Practical solution techniques for First-Order MDPs. *Artificial Intelligence*, *173*(5-6), 748–788.

Sanner, S., & McAllester, D. (2005). Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1384–1390.

Sanner, S., Uther, W., & Delgado, K. (2010). Approximate dynamic programming with affine ADDs. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*.

Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Muller, M., Lake, R., Lu, P., & Sutphen, S. (2007). Checkers is solved. *Science*, *317*(5844), 1518–1522.

Schuurmans, D., & Patrascu, R. (2001). Direct value-approximation for factored MDPs. In *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 1579–1586.

Schweitzer, P., & Seidmann, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, *110*(2), 568–582.

Sen, P., Deshpande, A., & Getoor, L. (2008). Exploiting shared correlations in probabilistic databases. In *VLDB*, pp. 809–820.

Sen, P., Deshpande, A., & Getoor, L. (2009). Bisimulation-based approximate lifted inference. In *Proceedings of Uncertainty in Artificial Intelligence*, pp. 496–505.

Shavlik, J. (1989). Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning*, *5*(1), 39–70.

Singla, P., & Domingos, P. (2008). Lifted First-Order belief propagation. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1094–1099.

St-Aubin, R., Hoey, J., & Boutilier, C. (2000). APRICODD: approximate policy construction using decision diagrams. In *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 1089–1095.

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. In *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning*.

Teichteil-Koenigsbuch, F., & Fabiani, P. (2006). Symbolic stochastic focused dynamic programming with decision diagrams. In *Proceedings of the Fifth IPC at ICAPS*.

Teichteil-Koenigsbuch, F., Infantes, G., & Kuter, U. (2008). RFF: A robust FF-based mdp planning algorithm for generating policies with low probability of failure. In *Proceedings of the Sixth IPC at ICAPS*.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, *8*(3-4), 257–277.

Tsitsiklis, J., & Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, *22*(1-3), 59–94.

van Otterlo, M. (2008). *The logic of Adaptive behavior: Knowledge representation and algorithms for adaptive sequential decision making under uncertainty in First-Order and relational domains*. IOS Press.

Veloso, M. (1992). *Learning by Analogical reasoning in general problem solving*. Ph.D. thesis, Carnegie Mellon University.

Walker, T., Torrey, L., Shavlik, J., & Maclin, R. (2007). Building relational world models for reinforcement learning. In *Proceedings of Inductive Logic Programming*, pp. 280–291.

Wang, C. (2008). *First-Order Markov decision processes*. Ph.D. thesis, Tufts University.

Wang, C., Joshi, S., & Khardon, R. (2007). First-Order decision diagrams for relational MDPs. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 1095–1100.

Wang, C., Joshi, S., & Khardon, R. (2008). First-Order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, *31*(1), 431–472.

Wang, C., & Khardon, R. (2007). Policy iteration for relational MDPs. In *Proceedings of Uncertainty in Artificial Intelligence*.

Weld, D., Anderson, C., & Smith, D. (1998). Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 897–904.

Yoon, S., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 352.

Younes, H., Littman, M., Weissman, D., & Asmuth, J. (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, *24*(1), 851–887.