Planning for Welfare to Work

Liangrong Yi and **Raphael Finkel** and **Judy Goldsmith**
Department of Computer Science
University of Kentucky
Lexington, Kentucky 40506-0046

Abstract

We are interested in building decision-support software for social welfare case managers. Our model in the form of a factored Markov decision process is so complex that a standard factored MDP solver was unable to solve it efficiently. We discuss factors contributing to the complexity of the model, then present a receding horizon planner that offers a rough policy quickly. Our planner computes locally, both in the sense of only offering one action suggestion at a time (rather than a complete policy) and because it starts from an initial state and considers only states reachable from there in its calculations.

Introduction

A naïve plan for building a decision support system might be to first model the decision-making process in a well-understood, mathematically correct model, and then use that model and off-the-shelf planning algorithms to suggest optimal plans, or evaluate user-suggested plans. However, this approach often requires replanning due to the changing world.

This paper presents a decision-support opportunity, a place where the naïve plan failed, and an approximation developed in response to that failure. The fact that this approximation actually addresses some of the concerns introduced with simplifications of the original model is a bonus for this approach.

The underlying decision-making we study is that of social welfare case managers in the Welfare to Work program. We begin with a brief introduction to Welfare to Work.

The “Welfare to Work” Program

The Welfare to Work (WtW) government program aims to return welfare recipients to the work force. In order to do so, clients—welfare recipients, most of them single mothers—are required to participate in activities intended to make them “work ready”. In addition, a life-time limit of 60 months of benefits is imposed on each individual. Those 60 months are not required to be continuous. The activities and services include education, from basic life skills to graduate work, including specific job-skills courses and courses on how to hunt for jobs. WtW also provides health benefits, various forms of counseling, childcare and transportation.

Typically, a client who enters the program has a series of interviews and takes various tests of interest and ability, and is assigned to a case manager. The case manager helps the client plan which activities to complete and which services to access. The plan should optimize the expected reward to the client. Given the uncertainties about the client’s current situation, her compliance with the plan, and the effects and durations of her activities, the plan is renegotiated regularly. The plan is also frequently affected by changing laws and regulations, and availability of services (Dekhtyar *et al.*, 2005).

Case managers usually have a heavy case load. It is hard for them to quickly follow the client’s status and keep up with changing regulations and availability of services. A decision support system could help them greatly.

Planning for WtW

In this section, we discuss the mathematical structure of our model and show a common method to plan optimally over such a model.

Modeling WtW as a Factored MDP

An MDP is a four-tuple $\langle S, A, Pr, R\rangle$. S is a finite state space. A state $s \in S$ is a description of the client at a particular time. A is a finite action space. The state transition function $Pr(j|i, a)$ defines the probability that the client moves from state i to state j by taking the action a. The reward function R assigns a value to a state s, giving the utility for that state. As we understand from the anthropologists who have interviewed the case managers, case managers do not calculate the cost of services to clients. The only cost they deal with is that they strive to keep a certain percentage of the clients “in compliance” with strict regulations about percentage of time spent in volunteer or paid work. Therefore we assume that actions have no cost. A policy is a mapping from states to actions, specifying which action to take in each state. To solve an MDP is to find a policy that optimizes the long-term expected reward of each state, or of the start state.

States can be described explicitly or in a factored representation. The explicit representation enumerates all the states directly. The factored representation defines a set of attributes that are sufficient to describe the states (Boutilier, Dean, & Hanks, 1999). Factored representations are usually more convenient and compact. Because of the large domain of WtW, we pick factored MDPs to model it.

A client's situation is modeled by a state, services are modeled as actions, and client preferences as reward functions (Dekhtyar *et al.*, 2008). Regulations and limitations of clients are constraints. The attributes are client characteristics, such as the client's age, education level, number of children, literacy level, confidence, commitment level, and so on. An action reflects client participation in one of the activities or services. A list of actions is shown in Table 1. Taking an action may affect some of the client's characteristics. For example, ATTENDING COMMUNITY COLLEGE can improve the client's confidence. Each client has her own expectation for joining the program. One wants to improve her literacy level, while another wants to gain vocational skills. Those preferences are represented as reward functions in our model.

Table 1: The List of Action

$ActionName$	$ActionName$
Job and Post-Sec. Education	ESL
Short Term Training	Volunteer Placement
Vocational Rehabilitation	Job Readiness Class
On Job Training	Literacy Training
College	Community Service
Adult Basic Education	Community College
High School	Vocational School
Group Job Search	

There is uncertainty in the client's current situation. It is very reasonable to model the domain as a partially observable MDP (POMDP). But that would make the computation more complicated. Thus we assume the model is fully observable.

A General Method to Solve Factored MDPs

Two general methods to solve MDPs are value iteration and policy iteration. They are both dynamic programming techniques. In value iteration, a policy is considered to be optimal when the following formula reaches its fixed point (Bellman, 1957).

$$V^*(s) = R(s) + \max_a[\gamma \sum_{s' \in S} Pr(s'|s,a))V^*(s')],$$

where $V^*(s)$ is the optimal total expected value function and γ is the discount (a real number between 0 and 1).

In a factored domain, states may be described by the values of the attributes. State transition probabilities are represented by dynamic Bayesian networks (DBN) (Pearl, 1988). A DBN describes an action. From the DBN, we can clearly know how the current state affects the next state when the action is executed. For each affected attribute, a conditional probability table (CPT) stores the transition probabilities. CPTs can be represented by algebraic decision diagrams (ADD) (R.I. Bahar *et al.*, 1993). ADDs can also describe value functions and policy. Jesse Hoey *et al.* propose **stochastic planning using decision diagrams** (SPUDD) (Hoey *et al.*, 1999). SPUDD is an algorithm for finding optimal or near-optimal policies for factored MDPs. The algorithm is an extension of value iteration that uses decision diagrams.

We have built a planner based on SPUDD, with minor modifications to match the specific features of WtW. Unfortunately, the planner failed to generate a policy in an acceptable amount of space (gigabytes) and time (hours).

Why It's So Difficult

Why does SPUDD fail in the WtW domain? We analyze the reasons in this section.

Large Domain

The WtW domain is large. It requires 50 attributes to represent a state. Many attributes have more than three possible values. The attribute LENGTH OF EMPLOYMENT can be MORETHANTWOYEARS, SIXMONTHS, ONEYEAR, ONEYEARSIXMONTHS and TWOYEARS.

A full policy assigns an action to each state. Therefore the large number of attributes also results in a large policy and a value function defined over a large domain.

The number of actions also affects the performance of MDP planning. The more actions, the more choices we need to consider. The WtW model has 15 actions, enough to make the computation very slow.

Action Complexity

To specify an action for SPUDD input, we must give a CPT for each attribute. Those information comes from the case managers' experiences based on interviews the anthropologists conducted with the case managers and on the case managers' input via elicitation software designed for this project (Mathias, Goldsmith, & Dekhtyar, 2008). The CPTs are represented in ADDs. Larger CPTs can lead to larger ADDs, and therefore to slower planning.

The execution of an action affects a set of attributes; let's call them informally the "output attributes" of that action. Those output effects depend on the current values of a set of attributes; let's call those the "input attributes" of the action. For example ATTENDING COLLEGE can improve the client's educational level, so educational level is an output attribute of this action. Similarly, the client's employability is an output attribute of ATTENDING SHORT TERM TRAINING. VOLUNTEER PLACEMENT affects many output attributes, including APTITUDE, GOAL, and INCOME. INCOME is very seldom an input attribute, but it is often an output attribute. By way of contrast, CONFIDENCE and SKILLS are often input attributes, but are seldom output attributes of actions.

Our full WtW model involves about 50 attributes. Actions seldom have many output attributes. For example, LITERACY TRAINING can improve a client's LITERACY LEVEL but won't change her CRIMINAL RECORD. Since any action affects only a few attributes, it leaves many attributes unchanged. The values of those attributes are deterministic and not dependent on the action. As a result, most parts of the descriptions of actions are the same. Redundancy is part of the explanation for the large size of our action ADDs.

We implemented SPUDD on Pascal Poupart's ADD package (Poupart, 2007). The SPUDD input parser of that package requires mention of every attribute for each action, even those that are not output attributes for the action.

In fact, very few attributes are output attributes for any action. A few are output attributes for almost every action: CONFIDENCE, DEPRESSION, LITERACY LEVEL, and WORK READINESS. Let's consider the situation that the preference involves only on a single attribute. If it is not an output attribute of only a few actions, the planner converges quickly. If it is an output attribute for most actions, computation converges much more slowly.

Large Value Functions

We consider clients' preferences as reward functions. Preferences describe the client's desires about the attributes. If a client's main purpose is to get a job, she might have an opinion about HIGH LIKELIHOOD THE CLIENT WILL STAY EMPLOYED. She can specify her desire for this attribute as one of five degrees: MUST HAVE, PREFER, INDIFFERENT, OPPOSE and CAN'T HAVE. If the state indicates that there is a high likelihood that she will stay employed, she accrues a reward of 1, 0.5, 0, -0.5, or -1, depending on which degree she chooses.

A preference can be simple or combinational. A simple preference contains only one attribute; a combinational preference consists of at least two attributes.

Let's look at a simple example. A client enters the program. She is pessimistic and lacks interest. The first step to help her might be to lift her confidence and interest. We represent this situation by two independent preferences: *the client must have a high level of confidence* and *the client's interest is preferred to be true*. Each preference is represented by an ADD, as shown in Figures 1 and 2. The preferences are used to define the MDP reward functions. Summing up the two ADDs, the final version of the initial reward function is shown in Figure 3.

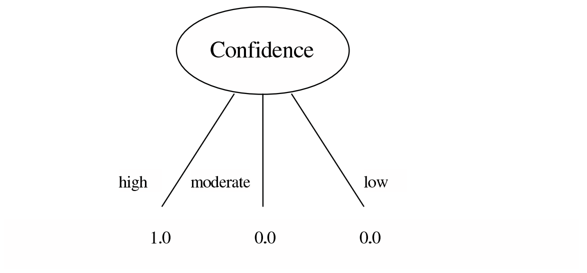

Figure 1: A Reward Function with Preference on the Attribute CONFIDENCE

Although the reward function might be very simple, the planning process can lead to a monstrously large value function. SPUDD performs classical value iteration on values stored as ADDs. At each step of the value iteration, it needs to multiply each action ADD (usually with n original variables and 1 primed variable, where n is the number of attributes) with the most recent value function ADD (up to n primed variables). At some intermediate steps, this calculation might generate an ADD with $2n$ variables. The state space of the WtW domain is huge, due to the large number of attributes. The transition probabilities in CPTs can vary a lot. Those two facts result in a dramatically growing value function ADD. Since many states have different values, the ADD is not significantly smaller than a full decision tree; it has a tremendous number of leaves. The advantage of compact ADDs disappears completely. After a few steps of value iteration, the value function ADD reaches a size beyond our memory capacity. We are building our own ADD package and hope it will be more efficient. However, simplifying the SPUDD input won't reduce the size of the reward function. We observe that benchmark models used by the community have extremely simple utility functions that do not adequately represent the complex preferences of WtW clients.

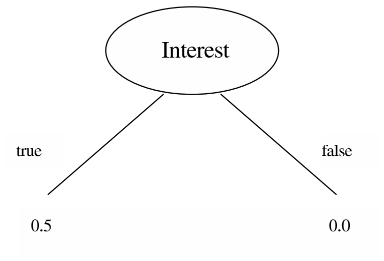

Figure 2: A Reward Function with Preference on the Attribute INTEREST

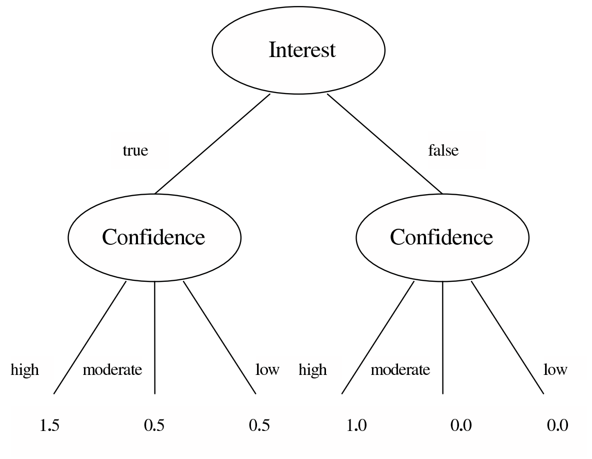

Figure 3: The Sum of the Two Reward Functions

To combine leaves with slightly different values, we limit the digits of precision on intermediate results. The probabilities in CPTs and the other parameters have one digit after the decimal point. Truncation is applied at all ADD operations so that the intermediate and the final values have up to two digits after the decimal point. The size of the value function ADD shrinks. But after each iteration the number of possible numerical values still goes up. The value function ADD is still problematic even after the truncation operation.

The Receding Horizon Planner

In this section we introduce the basic idea of a receding horizon planner and its design.

Analysis of WtW Domain

Given that SPUDD fails to generate a full policy for all possible states, we have to find a compromise. We may either reduce the model or relax the requirements of planning.

MDP planning computes the values of all the states and finds optimal actions for them. Is such global scrutiny really needed in WtW? When a client consults a case manager, she is in a particular state. For example, a 30-year-old woman with a 4-year old son has completed high school. Her YEARS OF SCHOOLING attribute won't be changed to REMEDIAL or POST-SECONDARY. And the AGE attribute won't take the value of TEENAGE in her case. For one client, it is impossible to reach all the states in the state space. The unreachable states won't affect our decisions at all. We should only focus on the possibly reachable states.

Do we need a 60-month plan when the client enters the program? Such a plan is impractical and unrealistic. The regulations change, the availability of services changes, and the client's preferences change. It's not wise to spend a lot to compute a long-term plan which will suffer many modifications in the future. When a client comes to the case manager, what she cares about is what to do next. What happens in three years is not immedeately important to her. Further, predicting the future is inaccurate. A client might be ineligible to access a service because she has only one child. But she might be able to take the service 20 months later if the rule changes or she has a new baby.

In WtW, the clients meet with the case managers regularly. They discuss the client's preferences, analyze the client's progress in the program, and decide on services to take next. A planner that can give suggestions for the next step or the next several steps, based on the client's current situation, will meet most of their requirements.

Overview of the Receding Horizon Planner

The fact that a single client won't explore the whole state space lets us take advantage of reachability. Reachability can be used to reduce the size of the MDP and make planning easier (Boutilier, Brafman, & Geib, 1998). We focus on the client's current state and extend to the possible successive states step by step. Reachability can help us to exclude a large number of states and make the problem much simpler.

We consider two strategies for updating the values: synchronous and asynchronous. Classical value iteration is synchronous; it evaluates the values of all the states at the same time. Asynchronous methods update states one at a time (Smith & Simmons, 2006) (Dai & Goldsmith, 2007). Since we expand the state space step by step, we choose to update the values in an order. The 0-step-to-go (newly expanded) states are updated first, then the 1-step-to-go, and so on.

The notion of receding horizon planning is not new. It is also called *model predictive control* (MPC). MPC has been widely implemented in industry (Morari & Lee, 1999). The basic idea is iterative on-line (also called real-time) optimization. It includes a process model and process measurements (Nikolaou, 1998). An on-line calculation is used to find a t-step optimal strategy for the current state. The first step is implemented and feedback is collected from process measurements. Then the optimization calculation is repeated on the new current state. We bring this idea to WtW planning. The planner computes a t-step optimal policy for the current state. The client takes the first action in the policy. The client then enters a new state, and her expectations about the effects of the WtW program may have changed. Based on the new information, the next round of planning is executed on the client's new state.

Due to the complicated model, it is impractical to compute a big policy (when t is large) at each step. Horizon control is used to reduce the policy size. The upper bound for the planning time can be set according to the case manager's requirement. When computation overruns the time limit, it stops and releases a partial policy. We let feasibility dictate the planning horizon. Our limited horizon can obscure long-term effects, but long-term planning in WtW is a dicey business. Horizon control guarantees the progress of planning in an acceptable time.

We have built a receding horizon planner. It starts from an initial state representing the client's current situation and expands the state space to states that are t-step reachable from the initial one. The output is a partial policy for the initial state and the expanded states.

More about the Planner

The receding horizon planner executes a number of iterations. Each iteration includes two phases: expanding the state space and updating the value functions. The algorithm is shown as follows. The notation $s \rightarrow s'$ means that s' is a possible next state after s when the action under consideration is taken.

```
Receding-Horizon-Planner()
$S[0] \leftarrow s_0$, $V[0] \leftarrow R[s_0]$
$i \leftarrow 0$
repeat
    $S[i+1] \leftarrow$ State-Space-Expansion($S[i]$)
    $i = i + 1$
    Update ($S$, $V$, $i$)
until time of Update($S$, $V$, $i$) $\geq limit$
End Receding-Horizon-Planner
```

```
State-Space-Expansion($S$)
for each $s \in S$ do
    for each action $a \in A$ do
        apply $a$ on $s$
        if $s \rightarrow s'$ and $s' \notin S$ then
            add $s'$ to $S$
        end if
    end for
end for
return $S$
End State-Space-Expansion
```

```
Update ($S$, $V$, $i$)
for each $s \in S[i]$ do
    $V[i](s) \leftarrow R(s)$
end for
while $i > 0$ do
    for each $s \in S[i-1]$ do
        $V[i-1](s) \leftarrow R(s) + max_{a \in A}\{\sum_{s'} Pr(s'|a,s)V[i](s')\}$
    end for
    $i = i - 1$
end while
End Update
```

The state space expansion subroutine enlarges the state space a little bit at each step. The value function update subroutine computes the values for the states included in the state space. The calculation is similar to finite horizon planning. The newly expanded states have 0 step to go, and their values are their immediate rewards. The values of states from the previous step are updated, using the values of the new states that they can reach.

Let's go through the first iteration. The initial state space has only one state: the current state of the client, s_0. We call that state space $S[0]$. The state space expansion subroutine expands the state space by applying all applicable actions a to the initial state. The expanded state space is $S[1]$. $S[1]$ contains the initial state and the states that are one-step reachable. The newly added states form the fringe of the state space. The next phase is to update the value functions. We use $V[0]$ to represent the value functions for the states in $S[0]$. At each iteration, the value functions on the fringe are assigned according to the reward function R. Now the value functions $V[1](s)$ for states s in $S[1]$ are equal to $R(s)$. Given $V[1]$, we can update $V[0]$:

$$V[0](s) = R(s) + max_{a \in A}\{\sum_{s'} Pr(s'|a,s)V[1](s')\}$$

At this point, $V[0]$ represents a horizon-one value function for $S[0]$, from which we can extract a one-step policy. If we can compute $V[0]$ quickly enough, we carry out another iteration. We expand the state space one step further. $S[2]$ is the set of states reachable from $S[1]$, from which we update the value functions for $S[1]$ and then $S[0]$. We can now extract a two-step policy. The planner stops when it spends too much time computing the value functions. Then we get the policy for the initial state and the states that are a few steps reachable from the initial one.

This process of two iterations of state expansion and value update is also shown in Figure 4. In the figure, the solid arrows represent the first iteration and the dashed arrows represent the second iteration.

We can control the time required for planning by limiting updating time. Therefore, we can get a partial policy fairly promptly, so case managers and clients need not wait long.

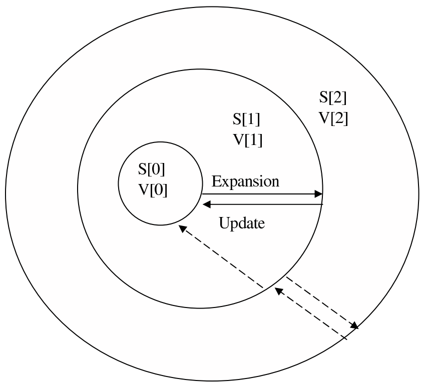

Figure 4: State Space Expansion and Value Update

Experiments and Results

We plugged the planner into our WtW software package. Information about the clients and services is already stored in the database. The client's preferences can be input to the database through an interface called PlanIt. PlanIt parses the input and calls the planner. The receding horizon planner computes a t-step policy and returns a suggested action based on the client situation and the preferences.

The planner can compute $S[10]$ and the related V values in about 10 seconds, so we can suggest an action that is optimal considering only the possible situations 10 steps in the future.

To evaluate the planner, we built a simulator. Given a client's state and her preferences, the planner suggests the best action to take. The simulator simulates the execution of that action: The simulated client reaches a randomly chosen new state based on the state-transition probabilities of the previous state and the suggested action. Once the simulator enters a new state, the planner is run afresh on that state. In this manner, a trial yields a trajectory of 10 states. We compute the expected total rewards for that trajectory, using a discount of 0.9, which treats later rewards as less important. We ran 100 such trials for each of 3 {starting state, client preference} pairs to compute the average expected total rewards. We also varied the farthest horizon t in planning. The result is shown in Table 2.

Table 2: Average Expected Total Rewards

$CaseNo.$	$t = 3$	$t = 5$	$t = 7$	$t = 10$
1	14.4515	14.5239	14.6004	14.3189
2	6.0753	6.1993	6.1532	6.2043
3	8.6112	8.6314	8.6557	8.6100

From the table, we can see that the resultant values from different planning horizons are very close. For each client state, only a very small number of iterations (probably 3) is required to get a good policy. Thus the receding horizon planning is very efficient in the WtW domain.

We didn't compare recommendations under different horizons. We note that the "best" action is not unique. Different actions may have the same effect. For example, “College Study” and “Community College” can raise a client’s confidence with the same probabilities. Even if plans recommend different actions, they might have the same expected effects; instead, we compare the expected total reward.

Conclusions

Due to the complexity of the WtW domain, the general method of MDP solving is not efficient. But after a clear analysis of the requirements and some simplifications, we have a working planner that satisfies the basic requirements of the domain. The two characteristics of the new planner are consideration of reachability and online optimization. A client won’t get to all the states in the state space, so the planner only concentrates on the states that are reachable from the client’s current state. The planner does not execute classical value iteration on the reachable states. Instead, it only computes a t-step optimal policy. The client takes the first suggested action and then we plan for her again.

When a client consults the case manager, she is in a particular state (the initial state in the receding horizon planner). What she and the case manager care about is what she should do next. Our planner can generate a suggestion of what action she needs to take. We think that is exactly what one expects a planner to provide. The regulations of the welfare programs and the available services change frequently, but the receding horizon planner, with its online optimization, is untroubled by changing environments. We believe that the receding horizon planner can satisfy the basic requirements of the Welfare to Work program and could be used to supplement or reinforce the case managers’ hard-earned experience.

References

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Boutilier, C.; Brafman, R.; and Geib, C. 1998. Structured reachability analysis for Markov decision processes. In *the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 24–32.

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

Dai, P., and Goldsmith, J. 2007. Topological value iteration algorithm for markov decision processes. In *the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1860–1865.

Dekhtyar, A.; Finkel, R.; Goldsmith, J.; Goldstein, B.; and Mazur, J. 2005. Adaptive decision support for planning under hard and soft constraints. In *AAAI Spring Symposium on Decision Support in a Changing World*, 17–22.

Dekhtyar, A.; Goldsmith, J.; Goldstein, B.; Mathias, K. K.; and Isenhour, C. 2008. Planning for success: The interdisciplinary approach to building bayesian models. *International Journal of Approximate Reasoning, to appear*.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *the Fifteenth Conference on Uncertainty in Articial Intelligence (UAI-99)*, 279–288.

Mathias, K. K.; Goldsmith, J.; and Dekhtyar, A. 2008. Bayesian construction: from qualitative to quantitative nets.

Morari, M., and Lee, J. 1999. Model predictive control: Past, present and future. *Computers and Chemical Engineering* 23:667–682.

Nikolaou, M. 1998. *Model Predictive Controllers: A Critical Synthesis of Theory and Industrial Needs*. Advances in Chemical Engineering Series. Academic Press.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo.

Poupart, P. 2007. personal communication.

R.I. Bahar; E.A. Frohm; C.M. Gaona; G.D. Hachtel; E. Macii; A. Pardo; and F. Somenzi. 1993. Algebraic decision diagrams and their applications. In *IEEE /ACM International Conference on CAD*, 188–191.

Smith, T., and Simmons, R. 2006. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *the Twenty-First AAAI Conference on Artificial Intelligence (AAAI-06)*, 1227–1232.