Time Complexity of the Incremental-Pruning Algorithm for Markov Decision Networks

H.H.L.M. Donkers J.W.H.M. Uiterwijk H.J. van den Herik
{donkers,uiterwijk,herik}@cs.unimaas.nl
Department of Computer Science, matriks-cs
Universiteit Maastricht
P.O. Box 616, 6200 MD Maastricht, The Netherlands

Abstract

A new representation technique for the planning of decisions under uncertainty is presented. The technique, called Markov decision networks (mdns), solves planning tasks that cannot or can hardly be solved by any other known representation technique, such as decision trees, in‡uence diagrams, and pomdps.

We show that linear mdns are pomdps. Hence, the incremental-pruning algorithm for pomdps is applicable to this subclass of mdns. It turns out that the time complexity of the algorithm is lower for mdns than for general pomdps, due to the division of the state space into state groups. An example shows the practical bene…t of our …ndings: an optimal strategy can be computed in a few seconds.

Contents

1. Introduction

In Artificial Intelligence, the planning of decisions for agents in uncertain environments is a main topic. The treatment of the subject is diverse. Depending on the kinds of decisions, agents, and uncertainty, a whole range of representation techniques is available to model and solve the planning tasks [8]. In this paper we concentrate on those planning problems for which (1) the uncertainty is modelled by Bayesian probabilistics, (2) there is one single agent that must build an optimal decision plan, (3) all decision options are completely known to the agent, and (4) the quality of a plan is measured only by its expected pay-off.

At present, there are several techniques available to deal with the above type of decision problem, but none of them is capable of solving all decision problems so typed satisfactorily. The oldest technique is the classical decision tree. It can only be used in small instances for which an exhaustive search through all possible plans is feasible. The influence diagram, in its original form, has as drawback that the ordering of the decision options is fixed by the structure of the diagram, and that repetition of decisions is inhibited. There are some alterations of the influence diagrams that try to alleviate these shortcomings, but full flexibility is still not reached. The technique of partially observed Markov decision processes (pomdps) offers the highest possible flexibility of combining decision options in any order. It confines the probabilities involved to behave Markovian, but it lacks the clear semantics of the influence diagrams. Moreover, even the most efficient exact solution method for pomdps (i.e., incremental pruning) is only feasible for small instances [10].

In this paper we introduce a new technique for solving a special type of decision problem: the Markov decision network (mdn)[1]. As the name indicates, we use a Markovian probability model, which gives the technique the same kind of planning flexibility as the pomdp technique does. However, we combine this flexibility with a graphical representation and understandable semantics, both improving the capability to model decision problems. The demands imposed on the decision problems to be modelled by an mdn are severe. But once these demands have been complied with, mdns offer a clear representation and a set of efficient solution methods.

After introducing the framework for mdns and two general solution methods, we show that linear mdns, a subclass of mdns, are pomdps. This means that for these mdns, solution methods for pomdps are applicable. Especially the incremental-pruning algorithm seems to be appropriate. The particular structure of mdns enables us to improve this algorithm for mdns.

The time complexity of this improved algorithm is investigated and we show it to be lower than that of the general algorithm for pomdps. To conclude this paper, we give a small example of an mdn and its solution obtained by incremental pruning.

[1] Part of this paper has been submitted for publication elsewhere ([5]).

2. Markov decision networks

The concept of mdns originates from the same source as the influence diagram, namely from the decision tree. The observation that different end nodes in a decision tree often represent the same final states of the world and that sometimes even whole subtrees are repeated in a decision tree, led Scott Olmsted in 1983 to the introduction of the concept of coalescence in decision trees [7]. An analogous observation in decision trees for medical use resulted in the introduction of the concept of mdns [2]. The idea of an mdn representation also stems from the first author's observation at the Department of Medical Informatics that medical students often have difficulties to understand and set up decision trees. Both observations motivated our group to simplify the concept of decision trees in the direction of compressing the tree into a network [2]. The idea was to provide the students with a straightforward technique to model and solve (medical) decision tasks. Of course, we were aware that such a simplification would reduce the class of decision tasks that could be handled. Later it transpired that the new concept was pivotal for solving planning tasks automatically, which is our present focus of research. Below we present a concise formal framework for mdns. The complete framework, along with the necessary proofs, is given in [3].

2.1. A framework for MDNs

An mdn is a quintuple of sets $\langle A; E; Z; W; G\rangle$, where A is the set of available actions $a_1; a_2; :::$, and E the set of available experiments $e_1; e_2; :::$. Actions change the state of the world, experiments give uncertain information on the actual state of the world by producing some outcome. For every experiment e_i, a non-empty set Z_{e_i} of outcomes (or results) $z_{e_i;1}; z_{e_i;2}; :::$ exists. The set Z is the union of these outcome sets Z_{e_i}. The results of an action or an experiment only depend on the actual state of the world. This property, the Markov property, gave its name to the Markov decision network.

The states of the world are represented by the finite set $W = \{w_1; w_2; :::\}$. The set $G = \{G_1; G_2; :::\}$ is the set of state groups, where $G_i \subseteq W$. These state groups form a partition of W. Two special state groups are the start group and the end group, represented by G_0 and G_u[1], respectively. The start group G_0 is the group in which the world is at the start of a plan, the end group G_u is the group that must always be reached when the plan ends and in which the final rewards (utilities) are received. It is possible to have a state group that is both start group and end group.

Three functions G^{in}, G^{out}, and G^{test} determine the structure of an mdn: function $G^{in}(a)$ gives the state group for which action a is meaningful, function $G^{out}(a)$ gives the state group that has been reached after action a has been performed, and function $G^{test}(e)$ gives the state group for which experiment e is meaningful. The inverse functions, A(G) and E(G), give the actions and experiments that are available for group G.

The probabilities in an mdn are defined by: (1) for each action a in A: a $|G^{out}(a)| \times |G^{in}(a)|$ transition probability matrix A_a, where $A_a(i; j)$ is the probability that state i in group $G^{out}(a)$ will be reached if the world was in state j of group $G^{in}(a)$ just before the action was performed; (2) for each experiment e in E: a $|Z_e| \times |G^{test}(e)|$ sensitivity matrix E_e, where $E_e(i; j)$ is the probability that the i-th outcome of Z_e is observed by the agent when the world is in state j of group $G^{test}(e)$; and (3) for the prior probabilities: a probability vector x_0, a $|G_0|$-sized vector in which $x_0(i)$ denotes the prior probability of the world being in state i of group G_0:

As a result of these probabilities, the state groups are mutually exclusive with respect to the probability distribution over the states. If there is a positive probability of the world being in one state of a group, then there is zero probability of the world being in a state of another group.

The costs in an mdn are given by the two functions K(a), the cost of doing action a, and K(e), the cost of

[1] The subscript u in G_u refers to the fact that the states of the end group are connected to utilities.

performing experiment e. Utilities are expressed in a $|G_u|$-sized utility vector u in which u(i) 2 R denotes the utility of the world reaching …nal state i of end group G_u.

Decision plans are called strategies in mdns. Strategies can be represented by strings of action symbols a_i, experiment symbols e_i, and a special ending symbol 3. For instance, the strategy $s = \langle e_1; a_1; 3; a_2; 3 \rangle$ denotes the decision plan to start with experiment e_1 and to perform action a_1 if the …rst outcome of e_1 is observed, and a_2 if the second outcome of e_1 is observed. Strategies that can actually be realized in an mdn are called legal.

An episode is a sequence of events (actions and experiments) that could actually happen if the agent follows a legal strategy. Episodes can be represented by simple strings of symbols for actions and outcomes of experiments. (The string $h = \langle z_{e_1;1}; a_1 \rangle$ denotes the episode of experiment e_1 resulting in the …rst outcome, whereafter action a_1 is performed).

The (pay-o¤) value V (h) of an episode h is some function of the collected costs K (h) and the expected utility U(h) of the episode. The form of this value function V (h) can be chosen at will. However, any choice does in‡uence (or even determine) the solution method to be used for solving the mdn. The expected value V (s) of a strategy s is the expected (pay-o¤) value of the episodes that can happen if that strategy is followed:

$$V(s) = \sum_{h \in s} P(h)V(h) \qquad (2.1)$$

In this formula, P(h) is the probability that episode h occurs if strategy s is followed. Finally, the general task of solving an mdn is to …nd a legal strategy $s^¤$ which has the highest expected value $V(s^¤)$ among all legal strategies.

2.2. A graphical representation

The graphical representation of an mdn is an augmented directed graph. The graph has a node for every state group, represented by an oval. The names of the states in the group can be written inside the oval. If needed, the groups themselves can also be named. This group name is placed outside the oval. The start group and end group can be marked specifically. Actions are represented by directed arcs between the nodes. The names of the actions are written alongside the arcs. Finally, experiments are represented as triangles that are linked to the oval of their test group. The outcomes of an experiment are not represented in the graph.

Figure 2.1 gives an example of the graphical representation for an mdn that models a simple diagnostic planning task for repairing parts. In this case there are two state groups, each containing two states. There are also two actions (“Repair” and “Don’t Repair”) and two experiments (“test 1” and “test 2”). The planning task is to decide how to combine both tests in such a manner that a maximum number of parts is …xed while keeping the total costs as low as possible. On the base of this example we brie‡y discuss some extra semantics that mdns give rise to, in comparison with in‡uence diagrams and pomdps.

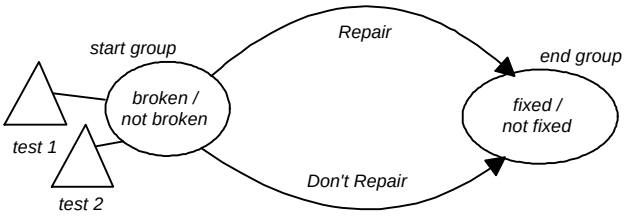

Figure 2.1: The graphical representation of an mdn.

2.3. Semantics

The meaning of a state group in mdns is broader than just an arbitrary set of states. State groups often denote speci…c views on the world that the agent uses at di¤erent moments in time. In the example of …gure 2.1, the …rst state group represents the view in which parts are either broken or not broken. The second group represents the view in which the agent looks upon parts as being …xed or not …xed. According to the graph, performing diagnostic tests is not meaningful in this stage and neither is reparation. Moving from one state group to another means not only changing the world, but also changing the perspective of the agent. Sometimes, the change of perspective is even the only alteration that is caused by an action. Hence, an action in an mdn means more than just having an e¤ect on the environment: it also changes, and sometimes exclusively, the inner state of the agent.

3. Solving MDNs for restricted strategy sets

Mdns were developed not only to model planning tasks, but also to provide a way of solving these planning tasks automatically. In this section, two solution methods for mdns are discussed that restrict the set of investigated strategies, viz. the standard decision-tree approach, and the decision-tree approach enriched with history rules.

3.1. Decision-tree approach

The decision-tree solution method is straightforward but restricted. It can be useful in many practical cases. In this approach, the mdn is used only as some kind of language through which a decision tree is constructed by the agent. The decision tree contains all plans that the agent judges to be relevant to investigate. The available decision options for the construction of the decision tree are derived from the arcs in the network. The branches of the chance nodes are constructed in accordance with the outcomes of the experiments.

After the construction of the tree, the quantitative information of the mdn is used to …ll in the probabilities and utilities of the decision tree automatically, and the classical folding-back procedure is used to solve the decision tree. In [2] a computer program is described that o¤ers this solution method in a user-friendly way.

3.2. History rules

The second solution method is an extension of the decision-tree approach. Instead of specifying a complete decision tree, the agent just states a set of rules that the generated plans should comply with. These rules are called history rules, because they should be able to decide whether a decision option is feasible given a history of actions and outcomes of experiments. Examples of history rules are: "never repeat an action" and "allow this action only after a positive test outcome." In [3, 4] an algorithm is presented that solves an mdn, given a set of history rules. It is imposed that these rules may not allow in…nite episodes, since then the algorithm will not end.

The history-rule algorithm handles four tasks in one run: it collects costs, computes probabilities, calculates expected values, and constructs an optimal strategy. In essence, it is the same as performing the folding-back procedure. The main di¤erence is that the decision tree itself is never constructed explicitly.

The complexity of this algorithm depends primarily on the history rules given. For instance, if one history rule would be "allow n subsequent repetitions of an experiment" then the complexity of the algorithm is at least $O(k^n)$, where k is the number of outcomes in the experiment.

4. Solving linear MDNs

The two methods for solving mdns discussed above restricted the number of plans under consideration. In this section, we introduce another solving method, incremental pruning, that considers all plans, but confines the value function $V(h)$ of the mdn to a linear one. The strength of this method compared to the former two methods is that not all plans have to be investigated to solve the mdn: an optimal strategy is constructed directly.

4.1. Linear MDNs as POMDPs

Mdns with a linear value function (which will be called linear mdns) are in fact a special case of partially observed Markov decision processes (pomdps). Linear value functions have the form $V(h) = \gamma U(h) - K(h)$. With this form it is possible to distribute the total pay-off value of an episode as partial rewards over the individual steps. At every step but the last one, the immediate reward is the negative cost of the action performed, or of the experiment performed, being $-K(a)$ or $-K(e)$, respectively. At the last step the reward is the (weighted) expected utility $\gamma U(h)$. The sum of these immediate rewards is the total pay-off value of the episode.

Because linear mdns are pomdps, algorithms for general pomdps are also applicable to linear mdns, although it might not always be such a good idea to do so. Algorithms for general pomdps are designed to deal with difficulties that occur often in normal pomdps but that are less frequent in mdns.

General pomdps (see [1, 6]) are characterized by three discrete and finite sets and three functions. The sets are: a set of states of the world W, a set of actions A, and a set of observations Z: The functions are: a state-transition probability function $P(w'|w;a)$, an observation probability function $P(z|w;a)$, and a real-valued reward function $r(a;w;w';z)$, with $a \in A$; $w;w' \in W$ and $z \in Z$ (The reward function can be reduced to a function $r^a(w)$ when rewards do not depend on the observation z or the next state w', as is the case in [1].)

In regard to general pomdps, linear mdns have four special properties: (1) separation of actions and observations, (2) fixed rewards, (3) partition of the state space, and (4) end-state utilities. These properties cause mdns to have a particular structure that can be exploited for efficiency. Below, each propriety will be explained in detail.

Separation of actions and observations – In standard pomdps every action is immediately followed by an observation. The mdn framework separates the actions from the observations: actions do not lead to observations and experiments do not change the state of the world. A linear mdn is converted into an equivalent pomdp by letting all actions be followed by a fixed single dummy-observation z_0 so that no information is gained by the actions ($P(z_0|w;a) = 1$), and by using a unit function as state-transition probability for every experiment ($P(w' = w|w;e) = 1$). Furthermore, the observations that can occur at an experiment are restricted to the possible outcomes of that experiment ($P(z|w;e) = 0$ if $z \notin Z_e$).

The separation of actions and experiments reduces the complexity of computations needed to obtain state probability vectors. In case of an action only a state transition matrix has to be applied; in case of an experiment only Bayes' rule has to be used.

Fixed rewards – The rewards that are obtained at each step in a general pomdp depend on the actual state, the action performed at that step, the observation obtained, and the next state. Often the rewards are a function of a subset of these four factors. The framework for mdns reduces the reward function to a minimum: the rewards at each step, except for the last one, only depend on the action or experiment performed at that step. They are equal to the negative costs of that action or experiment ($r(a;w;w';z) = r(a) = -K(a)$; and $r(e;w;w';z) = r(e) = -K(e)$).

Partitioned state space – The most apparent difference between mdns and general pomdps is the partition of the state space. The states in a mdn are partitioned into state groups, such that all probability mass is always concentrated in one group. Even more

important is the fact that the availability of every action and experiment is con…ned to one group.

End-state utilities – The pomdp "rewards" in mdns are de…ned as costs of actions and experiments. The real rewards are obtained after the …nal step when an end state is reached. These rewards are called utilities in mdns and the values of these utilities are determined by the end state. The fact that utilities can only be obtained in end states means that precisely those strategies that always lead to the end group are feasible.

4.2. Incremental pruning

An algorithm that …ts naturally to the kind of special properties of mdns described above, is the incremental-pruning algorithm of Zhang and Liu [1, 9]. The basic step in the derivation of this algorithm, according to [1], is to split the value function:

$$V^0(x) = \max_{a2A} \left(\sum_{w2W} r^a(w)x(w) + ° \sum_{z2Z} P(zjx;a)V(x_z^a) \right) \quad (4.1)$$

into simpler parts:

$$V^0(x) = \max_{a2A} V^a(x) \quad (4.2)$$

$$V^a(x) = \sum_{z2Z} V_z^a(x) \quad (4.3)$$

$$V_z^a(x) = \frac{\sum_{w2W} r^a(w)x(w)}{jZj} + ° P(zjx;a)V(x_z^a) \quad (4.4)$$

and to solve them separately:

$$S^0 = \text{purge}\left(\bigcup_{a2A} S^a \right) \quad (4.5)$$

$$S^a = \text{purge}\left(\bigoplus_{z2Z} S_z^a \right) \quad (4.6)$$

$$S_z^a = \text{purge}\,(f¿(®;a;z)j® 2 Sg) \quad (4.7)$$

$$¿(®;a;z)(w) = (1=jZj)r^a(w) + ° \sum_{w^0 2W} f®(w^0): P(zjw^0;a)P(w^0jw;a)g \quad (4.8)$$

The solution $V(x)$ of a pomdp is expressed in a set S of jWj-sized vectors $®$, each vector representing a separate strategy. In the formulas above, the notation x is used to denote the actual belief state, which is a probability vector de…ned on the states of the world, where $x(w)$ denotes the probability of the world being in state w. The notation x_z^a is used to denote the revised belief state after performing action a and observing z. The constant $°$ is a discount rate that is only used in in…nite horizon cases, otherwise $°$ is 1.

The function purge is an algorithm that reduces a given set of jWj-sized vectors to those vectors $®$ for which $®:x$ is maximal for some belief state x. The "incremental pruning" in a more strict sense is laid down in the solution of equation (4.6):

$$S^a = \text{purge}\left(\bigoplus_{z2Z} S_z^a \right) \quad (4.9)$$

$$´ \text{purge}(S_{z_1}^a © S_{z_2}^a © S_{z_3}^a © ::: © S_{z_{jZj}}^a)$$

$$= \text{purge}(:::(\text{purge}(S_{z_1}^a © S_{z_2}^a) © S_{z_3}^a)::: © S_{z_{jZj}}^a) \quad (4.10)$$

The operator © has the following meaning: if S_1 and S_2 are sets of vectors of equal length, then $S_1 © S_2$ is the set of vectors: $fs_1 + s_2 \mid s_1 2 S_1; s_2 2 S_2g$.

4.3. Specializing incremental pruning for MDNs

There are in fact three types of decisions in mdns: (1) the decision to perform an action a, (2) the decision to perform an experiment e, and (3) the decision to stop (d_i). The last decision is available whenever the world is in the end group G_u. Each one of the three types of decision has its own solution method: for stop-decisions, just the utility vector has to be used; for action-decisions, a transition matrix has to be applied; and for experiment-decisions, all outcomes have to be inspected. Hence, formula (4.2) can be rewritten into (4.11). Each of the partial solutions $V^u; V^a$; and V^e can be worked out separately according to the type of decision:

$$V^0(x) = \max \begin{cases} V^u(x) \\ \max_{a2A(G(x))} V^a(x) \\ \max_{e2E(G(x))} V^e(x) \end{cases} \quad (4.11)$$

Because the probability mass in a belief state x is always concentrated in one state group, it makes sense to use the expression G(x) to denote that state group. As a result, A(G(x)) and E(G(x)) form the decision options that are available at belief state x.

The solution for V^u is straightforward: the reward when stopping is the weighted expected utility $\mu : u : x$:

$$V^u(x) = \mu : u : x \tag{4.12}$$

At action-decisions, we just have to apply the appropriate transition matrix A_a to compute x^a_z in formula (4.4). The rewards $r^a(s)$ are laid down in the costs $-K(a)$.

$$V^a(x) = V(x^a_z) - K(a) = V(A_a x) - K(a) \tag{4.13}$$

For experiment-decisions, we need the following formula:

$$V^e(x) = \sum_z P(z|x;e) V(x^e_z) - K(e) \tag{4.14}$$

The posterior vector x^e_z can be computed by application of the sensitivity matrix E_e:

$$x^e_z = \frac{d(E^z_e) : x}{P(z|x;e)} \tag{4.15}$$

We use $d(E^z_e)$ to denote the diagonal matrix whose trace is equal to row z in matrix E_e. This notation seems awkward, but because V(x) is a linear function, the term P(z|x;e) will cancel out and only $d(E^z_e)$ remains, so $V^e(x)$ becomes:

$$V^e(x) = \sum_z V(d(E^z_e) : x) - K(e) \tag{4.16}$$

These formulas result in the following incremental pruning (ip) scheme:

$$S^0(G) = \text{purge}\left(S^d(G)\right) \tag{4.17}$$

$$S^d(G) = S^u(G) \oplus S^A(G) \oplus S^E(G) \tag{4.18}$$

$$S^u(G) = \begin{cases} \{\mu : u\} & \text{if } G = G_u \\ \{\} & \text{if } G \neq G_u \end{cases} \tag{4.19}$$

$$S^A(G) = \bigcup_{a \in A(G)} S^a(G) \tag{4.20}$$

$$S^E(G) = \bigcup_{e \in E(G)} S^e(G) \tag{4.21}$$

$$S^a(G) = \text{purge}(\{\alpha : A_a | \alpha \in S^{p(a)}(G^0)\}) - K(a) \quad (G^0 = G^{out}(a)) \tag{4.22}$$

$$S^e(G) = \text{purge}(\bigoplus_{z \in Z_e} S^e_z(G)) - K(e) \tag{4.23}$$

$$S^e_z(G) = \text{purge}(\{\alpha : d(E^z_e) | \alpha \in S^{p(e)}(G)\}) \tag{4.24}$$

(The notation $S^{p(:)}$ will be explained in the next section.) As is apparent from these formulas, a separate ip scheme exists for every state group G. In each step during the value-iteration process, this ip scheme can be performed for every group independently from the schemes of other groups. The complexity of an ip scheme is determined by the size of its state group, the number of actions and experiments attached to this group, and the number of outcomes that each experiment has.

4.4. Linking IP-scheme results

The ip scheme for a state group is linked to the results of ip schemes in previous value-iteration steps. In formula (4.22) a link is made to the ip scheme of the outgoing group $G^{out}(a)$ of action a through the expression $S^{p(a)}(G^{out}(a))$. A similar link is made through $S^{p(e)}(G)$ in formula (4.24).

The ip scheme can only be performed for a state group if that state group can have probability mass in the actual value-iteration step. At the first value-iteration step, for instance, only the end group G_u can have probability mass, because every strategy in an mdn should finish in the end group for all episodes. At the second value-iteration step only those state groups can have positive probability mass when the end group is reachable in one step through an action in the mdn graph. In general, at the n-th value-iteration step, only those groups can have positive probability mass, and therefore have their ip scheme performed, if the end group is reachable in exactly $n - 1$ steps in the mdn graph. If a link has to be made from one ip scheme to another in a previous step, the algorithm has to look for the most recent step in which the target ip scheme is available. In $S^{p(a)}(G^{out}(a))$, the function p(a) denotes the "previous" step, which is the most recent value-iteration step in which an ip scheme exists for the group $G^{out}(a)$.

```
MdnIncPrune (M = ⟨A, E, Z, W, G⟩, maxHor)
1.   S^0(·) ← ∅, S^0(G_u) ← {(d_·, ·, ·, u)}
2.   for h ← 1 to maxHor:
3.     S^h(·) ← ∅
4.     for all G ∈ G:
5.       p ← max(q | q < h, S^q(G) ≠ ∅)
6.       if p ≠ ? then
7.         S^h(G) ← purge(S^h(G) ∪ Ã(S^p(G), d_0))
8.         for all e ∈ E(G):
9.           S^0 ← purge({α·d(E_e^1)|α ⊕ S^p(G)})
10.          for z ← 2 to |Z_e|:
11.            S^0 ← purge(S^0 ⊗
                 purge({α·d(E_e^z)|α ⊕ S^p(G)}))
12.          S^h(G) ← purge(S^h(G) ∪ Ã(S^0, d_e))
13.      for all a ∈ A(G):
14.        G^0 ← G^out(a)
15.        p ← max(q | q < h, S^q(G^0) ≠ ∅)
16.        if p ≠ ? then
17.          S^0 ← purge({α·A_a|α ⊕ S^p(G^0)})
18.          S^h(G) ← purge(S^h(G) ∪ Ã(S^0, d_a))
19.    If S^h(G_0) ≠ ∅ and d = d_0
         for all decisions in S^h(G_0) then Stop.
```

Figure 4.1: Implementation of the incremental-pruning algorithm attempting to solve the mdn M in at most maxHor steps.

4.5. The incremental-pruning algorithm for MDNs

Figure 4.1 gives the pseudocode for an implementation of the incremental-pruning algorithm for mdns. The value-iteration steps are indicated here by the horizon variable h. A separate ip scheme is performed at every step for each state group. The results of the ip schemes for state group G at step h are represented by the set $S^h(G)$ of tuples $\langle d, \alpha \rangle$.

At line 7 a new, auxiliary decision is introduced: the wait-decision d_0. This decision has no meaning in the mdn itself, but is used in this algorithm to implement a stopping mechanism. The wait-decision is a decision to keep everything exactly the same for one step. The set $S^h(G)$ is filled at line 7 with the contents of the previous vector set for this group. The algorithm stops at line 19 when no decision can give a higher expected value than the wait-decision for any belief state in the start group G_0, which means that $S^h(G_0)$ only contains wait-decisions. The wait-decision has another effect: it is now only necessary to add a stop-decision to $S^0(G_u)$. The other stop-decisions that are mentioned in formula (4.19) of the ip scheme are replaced by wait-decisions added to $S^h(G_u)$. Function $\tilde{A}(S, d)$ at lines 7, 12 and 18 returns a set that contains all entries of S having the decision set to d and having α diminished with the costs of the decision d.

The purge routine called at lines 9, 11, 12, 17, and 18 is implemented by a slightly altered version of Lark's Filter algorithm as described in [1, 10]. The iterative purging process as described at line 11 is in fact the most simple incremental-pruning method possible and could well be replaced by one of the more efficient methods (see [1]).

5. Time complexity of the incremental-pruning algorithm for MDNs

We are curious to see whether a decrease of the time complexity is caused by the special structure of mdns. More speci…cally, we would like to investigate the role of the division of the state space into state groups. Hence, we are not aiming at obtaining a general formulation of the time complexity of the incremental-pruning algorithm for mdns,

Figure 5.1 provides the time complexity for each line of the algorithm given in …gure 4.1. The time complexities are based on the following assumptions:

- The number of value-iteration steps needed is at most $\hat{n}$.
- The size of all state groups is at most $\hat{s}$.
- The number of actions per group is at most $\hat{a}$.
- The number of experiments per group is at most $\hat{e}$.
- The number of outcomes per experiment is at most $\hat{r}$.
- The number of $\Re$-vectors produced per group per value-iteration step is at most $\Re$.
- The time complexity of the purge algorithm is at most $T(\text{purge}(\Re;\hat{s}))$.

From …gure 5.1 we can derive the following formula for the overall complexity:

$$\begin{aligned} T = {} & 1 + \hat{n}:[1 + |G|:[\hat{n} + 1 + \Re:(\hat{s} + 1) + Tp + \\ & \hat{e}:f(\Re:\hat{s} + Tp) + (\hat{r} - 1):(\Re:\hat{s} + \Re^2:\hat{s} + 2:Tp) + \\ & \Re:(\hat{s} + 1) + Tpg + \\ & \hat{a}:f2 + \hat{n} + \Re:\hat{s}^2 + Tp + \Re:(\hat{s} + 1) + Tpg] + \Re] \end{aligned} \tag{5.1}$$

(The symbol 'Tp' is an abbreviation of $T(\text{purge}(\Re;\hat{s}))$.) This formula can be rearranged by collecting all terms in $\hat{n}$, $\Re$, $\hat{s}$, and Tp:

```
1.   1
2.   n̂£ {
3.     1
4.     |G|£ {
5.       n̂
6.       1
7.       ℜ:(ŝ + 1) + T(purge(ℜ; ŝ))
8.       ê£ {
9.         ℜ:ŝ + T(purge(ℜ; ŝ))
10.        (r̂ - 1)£ {
11.          ℜ:ŝ + ℜ^2:ŝ + 2.T(purge(ℜ; ŝ)) }
12.        ℜ:(ŝ + 1) + T(purge(ℜ; ŝ)) }
13.      â£ {
14.        1
15.        n̂
16.        1
17.        ℜ:ŝ^2 + T(purge(ℜ; ŝ))
18.        ℜ:(ŝ + 1) + T(purge(ℜ; ŝ)) }}
19.    ℜ }
```

Figure 5.1: Complexity per line of the algorithm given in …gure 4.1.

$$\begin{aligned} T = {} & 1 + \hat{n} + \hat{n}:\Re + \\ & (\hat{n}:|G|)f\ [1 + 2:\hat{a}] + \\ & [1 + \hat{a}]:\hat{n} + [1 + \hat{e} + \hat{a}]:\Re + \\ & [\hat{e}:(\hat{r} + 1) + \hat{a} + 1]:\Re:\hat{s} + \\ & [\hat{e}:(\hat{r} - 1)]:\Re^2:\hat{s} + [\hat{a}]:\Re:\hat{s}^2 + \\ & [2:\hat{e}:\hat{r} + 2:\hat{a} + 1]:Tp\ g \end{aligned} \tag{5.2}$$

The square-bracketed terms in this formula are all bounded by:

$$\hat{d} = 2:\hat{e}:\hat{r} + 2:\hat{a} + 1 \tag{5.3}$$

So, the formula can be rewritten into:

$$\begin{aligned} T < {} & 1 + \hat{n} + \hat{n}:\Re + \\ & (\hat{d}:\hat{n}:|G|)f1 + \hat{n} + \Re + \Re:\hat{s} + \\ & \Re^2:\hat{s} + \Re:\hat{s}^2 + Tp\ g \end{aligned} \tag{5.4}$$

When the state groups in an mdn are of comparable size and each group has a comparable number of actions, experiments, and outcomes per experiment, then the following approximations can be made:

$$\hat{d} = \frac{D}{|G|} \tag{5.5}$$

$$\hat{s} = \frac{|W|}{|G|} \tag{5.6}$$

where D is de…ned by:

$$D = 2{:}|E|{:}\max_{e \in E} |R_e| + 2{:}|A| + 1 \tag{5.7}$$

This leads to the following formula for the time complexity:

$$\begin{aligned} T < {} & 1 + \hat{n} + \hat{n}{:}\alpha + \\ & (D{:}\hat{n}){:}\{1 + \hat{n} + \alpha + \alpha{:}\tfrac{|W|}{|G|} + \\ & \alpha^2{:}\tfrac{|W|}{|G|} + \alpha{:}\tfrac{|W|^2}{|G|^2} + T_p\} \end{aligned} \tag{5.8}$$

which …nally can be rewritten into:

$$\begin{aligned} T \in {} & [1 + \hat{n} + \hat{n}{:}\alpha + D{:}\hat{n}{:}(1 + \hat{n} + \alpha)] + \\ & \frac{[D{:}\hat{n}{:}(\alpha + \alpha^2){:}|W|]}{|G|} + \\ & \frac{[D{:}\hat{n}{:}\alpha{:}|W|^2]}{|G|^2} + [D{:}\hat{n}]{:}T_p \end{aligned} \tag{5.9}$$

On base of this formula, the time complexity of the incremental-pruning algorithm for mdns can be expressed as a function of the number of state groups, |G|:

$$T(|G|) = O(K_1 + \frac{K_2}{|G|} + \frac{K_3}{|G|^2} + K_4{:}T(\mathrm{purge}(\alpha, \frac{|W|}{|G|}))) \tag{5.10}$$

where $K_1 :: K_4$ are the terms that do not depend on the number of groups. The time complexity of the purge routine is a function of the number of vectors (α) and the size of the vectors ($|W|/|G|$). This function depends highly on the implementation of the Filter algorithm, but for the size of the vectors the time complexity will be more than linear.

To conclude, an increase in the number of groups |G| will either have no e¤ect on a term in expression (5.10) or will lower the value of a term. The total time complexity will therefore decrease if the number of state groups increases. Going from a general pomdp with only one state group to an mdn with multiple state groups will therefore decrease the total time complexity of the incremental-pruning algorithm.

6. Example

We will use a small example to show the working of mdns. Assume an agent has to go through 9 di¤erent subsequent stadia before reaching a …nal rewarding stadium. In every stadium the agent can be in one of two states (a good state and a bad one) but the actual state is unknown to the agent.

Going from one stadium to the next one, the agent can try to improve his situation by investing 25 units, having a probability of 30% to reach a good state from a bad state, but also having a probability of 20% to arrive in a bad state from a good one. The agent can decide to do nothing instead of investing, in which case he has only a probability of 10% to change his state from a bad one to a good on and also a probability of 10% to change from a good state into a bad one.

Investment actions are named a,b,c... and the do-nothing actions $a^0; b^0; c^0$,.... At every stadium, except the last one, the agent can use an experiment (named A, B, C, ...) with two outcomes to gain information on the actual state. The experiments cost 10 units each but are erroneous in 5% of the cases. The utility for reaching the bad state in the last stadium is 0, and is 1 for reaching the good state. The value function used is: $V(h) = \lambda U(h) - K(h)$, which is linear. The structure of the mdn that models this problem is given in …gure 6.1.

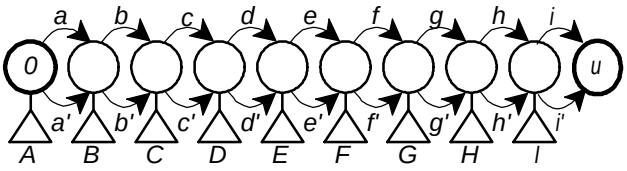

Figure 6.1: Graphical representation of the 10-stadia example.

Depending on the weight λ, the incremental-pruning algorithm …nds the following optimal strategies:

$s_1 : \{a^0b^0c^0d^0e^0f^0g^0h^0i^0\}$
$s_2 : \{a^0b^0c^0d^0e^0f^0g^0h^0\mathrm{I}\,i\}\,i^0\}$
$s_3 : \{a^0b^0c^0d^0e^0f^0g^0\{h\mathrm{I}\,i\}\,i^0\}\,h^0i^0\}$
$s_4 : \{a^0b^0c^0d^0e^0f^0\{g\{h\mathrm{I}\,i\}\,i^0\}\,h^0i^0\}\,g^0h^0\mathrm{I}\,i\}\,i^0\}$
$s_5 : \{a^0b^0c^0d^0e^0f^0\{g\{h\mathrm{I}\,i\}\,i^0\}\,h^0\mathrm{I}\,\mathrm{I}\,i\}\,i^0\}\,i^0\}\,g^0h^0\mathrm{I}\,i\}\,i^0\}\,i$

In s_2 we see, for example, the application of experiment I. It decides on a continuation with i or $\neg^0$. The solution sets of strategies that are optimal for some region of the prior belief space are for di¤erent levels of λ:

$\lambda = 1$:	$\{s_1\}$	$\lambda = 300$:	$\{s_2; s_3\}$
$\lambda = 200$:	$\{s_1\}$	$\lambda = 325$:	$\{s_3\}$
$\lambda = 250$:	$\{s_1; s_2\}$	$\lambda = 475$:	$\{s_3; s_4\}$
$\lambda = 275$:	$\{s_2\}$	$\lambda = 500$:	$\{s_3; s_4; s_5\}$

So, the higher the weight factor on the utilities, the more the agent is willing to invest in experiments. The understanding that investing is a good (even optimal) strategy, starts in the last stadium and progresses forwards in relation to an increase of the weight λ. The computations took at most 7 seconds on a Pentium II, 300 Mhz personal computer.

7. Conclusion

The new technique for modeling decision planning in uncertain environments, mdn, adequately combines the ‡exibility of pomdps with the semantics of in‡uence diagrams. Having shown that linear mdns are pomdps, we adapted the most e¢cient algorithm for solving pomdps known today i.e., the incremental pruning algorithm, to make it …t for linear mdns. We showed that the incremental-pruning algorithm adapted for mdns is more e¢cient in terms of time complexity than that for general pomdps.

Acknowledgement

The research is supported by the Dutch organization for air (Dutch: Lucht) and Water environment Information systems (lwi).

8. References

[1] Cassandra, A.R., Littman, M.L., and Zhang, N.L. (1997). Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes. Proceedings of the thirteenth conference on Uncertainty in Arti…cial Intelligence, pp. 54-61. Morgan Kaufmann Publ., San Mateo, CA. ISBN 1-55860-485-5.

[2] Donkers, H.H.L.M. and Hasman, A. (1988). Strategy, een Nieuwe Kijk op Beslisbomen. Proceedings of the 8st Dutch Medical Informatics Congress, pp. 65-69. VMBI, Utrecht, The Netherlands. ISBN 90-5005-022-0.

[3] Donkers, H.H.L.M. (1997). Markov Decision Networks. Technical Report CS 97-04. Universiteit Maastricht, Maastricht, The Netherlands.

[4] Donkers, H.H.L.M, Uiterwijk, J.W.H.M., and Herik, H.J. van den (1998). Markov Decision Networks. Submitted to ECAI'98.

[5] Donkers, H.H.L.M, Uiterwijk, J.W.H.M., and Herik, H.J. van den (1998). Solving Markov Decision Networks using Incremental Pruning. Submitted to UAI'98.

[6] Lovejoy, W.S. (1991). A Survey of Algorithmic Methods for Partially Observed Markov Decision Processes. Annals of Operations Research, 28(1-4), pp. 47-66. ISSN 0254-5330.

[7] Olmsted, S.M. (1983). On Representing and Solving Decision Problems. Ph.D. Dissertation Engineering-Economic Systems. Stanford University, Stanford, CA.

[8] Pomerol, J.C. (1997). Arti…cial Intelligence and Human Decision Making. European Journal of Operations Research, 99, pp. 3-25. ISSN 0377-2217.

[9] Zhang, N.L. and Liu, W (1996). Planning in Stochastic Domains: Problem Characteristics and Approximation. Technical Report HKUST-CS96031, Hong Kong University of Science and Technology.

[10] Zhang, N.L. and Liu, W (1997). A Model Approximation for Planning in Partially Observable Stochastic Domains. Journal of Arti…cial Intelligence Research, 7, pp. 199-230.