Finding the K best policies in a finite-horizon Markov decision process

Lars Relund Nielsen[*][†] Anders Ringgaard Kristensen

Department of Large Animal Sciences[‡]
Royal Veterinary and Agricultural University
Grønnegårdsvej 2
DK-1870 Frederiksberg C
Denmark

August 2, 2005

Abstract

Directed hypergraphs represent a general modelling and algorithmic tool, which have been successfully used in many different research areas such as artificial intelligence, database systems, fuzzy systems, propositional logic and transportation networks. However, modelling Markov decision processes using directed hypergraphs has not yet been considered.

In this paper we consider finite-horizon Markov decision processes (*MDPs*) with finite state and action space and present an algorithm for finding the K best deterministic Markov policies. That is, we are interested in ranking the first K deterministic Markov policies in non-decreasing order using an additive criterion of optimality. The algorithm uses a directed hypergraph to model the finite-horizon MDP. It is shown that the problem of finding the optimal policy can be formulated as a minimum weight hyperpath problem and be solved in linear time, with respect to the input data representing the MDP, using different additive optimality criteria.

Keywords: Finite-horizon Markov decision processes, stochastic dynamic programming, directed hypergraphs, hyperpaths, K best policies.

1 Introduction

Many decision problems are dynamic in nature and must be re-evaluated over time based on the state of some crucial underlying factors, e.g. machine state, company finances, etc. Often these problems can be modelled using Markov decision processes (*MDPs*) which have been widely used to model stochastic environments, due to their expressiveness and analytical tractability.

[*] Corresponding author.

[†] Present address: Research Unit of Statistics and Decision Analysis, Research Centre Foulum, P.O. Box 50, DK-8830 Tjele.

[‡] E-mail addresses: lars@relund.dk (L.R. Nielsen), ark@dina.kvl.dk (A.R. Kristensen).

MDPs model sequential decision-making problems. At a specified point in time, a decision maker observes the state of a system and chooses an action. The action choice and the state produce two results: the decision maker receives an immediate reward (or incurs an immediate cost), and the system evolves probabilistically to a new state at a subsequent discrete point in time. At this subsequent point in time, the decision maker faces a similar problem. The observation made from the system's state now may be different from the previous observation. The goal is to find a policy of choosing actions (dependent on the observation of the state) which maximizes the rewards after a certain time.

Finding an optimal policy for an MDP is a well studied topic. One of the first books on the subject was by Howard [12]. As the title[1] suggests the idea of the book was to combine the dynamic programming technique by Bellman [6] with the mathematically well established notation of a Markov chain. Since the publication of this book an intensive research in MDPs has been carried out.

A recent book on the subject summarizing results from the past decades is Puterman [27]. Here both finite and infinite horizon MDPs are considered, having either finite, countable or continuous state and action spaces. The optimality criteria considered are the expected total reward, the expected discounted total reward and the expected average reward.

Also *constrained* MDPs have gained considerably interest, see e.g. Altman [1], Altman and Shwartz [2]. Here the optimality criteria are maximized while keeping other type of costs below some given bound. Often these problems are solved using linear programs, if the system can be described by a finite number of states and actions. The linear program formulation of an MDP was introduced by Derman and Klein [9] and later further developed by Derman and A.F. Veinott [8], Hordijk and Kallenberg [11], Kallenberg [14]. For a survey on the subject see Kallenberg [15].

Hierarchic MDPs were introduced by Kristensen [17], Kristensen and Jørgensen [18]. It is a contribution to the solution of the problem referred to as the "curse of dimensionality", since it provides us with a way to reduce the state space of large problems. Recently, a standard software system for solving both MDPs and hierarchic MDPs has been developed by Kristensen [16].

Another research area that has been extensively studied in recent years is directed hypergraphs. Directed hypergraphs are an extension of directed graphs and undirected hypergraphs introduced by Berge [7]. Directed hypergraphs represent a general modelling and algorithmic tool, which has been successfully used in many different research areas such as artificial intelligence, database systems, fuzzy systems, propositional logic, and transportation networks. For a general overview on directed hypergraphs see Ausiello, Franciosa, and Frigioni [4].

The concepts of a hyperpath and a minimum weight hyperpath were introduced by Nguyen and Pallottino [19] and later the definition of a hyperpath in a directed hypergraph and a general formulation of the minimum weight hyperpath problem were given by Gallo, Longo, Pallottino, and Nguyen [10]. In general the problem is $\mathcal{NP}$-hard (see e.g. Ausiello, Italiano, and Nanni [5]), but for a special class of weighting functions polynomial algorithms exist (see e.g. Ausiello et al. [5], Gallo et al. [10] and Ramalingam and Reps [28]). For efficient data structures for storing hypergraphs see Nielsen [20] and Ausiello, Datri, and Sacca [3].

Recently, a general algorithm for finding the K minimum weight hyperpaths has been presented in Nielsen, Andersen, and Pretolani [24] and in Nielsen, Pretolani, and Andersen [21] the complexity of the algorithm is lowered. Algorithms for ranking solutions are useful

[1] *Dynamic programming and Markov processes*

since practical problems often include constraints which are hard to specify formally or hard to optimize. In that case the algorithm may be used to enumerate suboptimal hyperpaths until a hyperpath satisfying the hard constraint is found.

The study of directed hypergraphs has become an important aspect in finding optimal strategies/paths in stochastic time-dependent networks, Nielsen [20], Pretolani [26]. Here the travel time between two nodes is time-dependent, i.e. the travel time depends on the departure time from a node. Furthermore, it is assumed that for each departure time, the travel time may not be fully known and hence a probability function is used to express possible travel times. By using the K minimum weight hyperpaths algorithm from Nielsen et al. [21] it is possible to find the K best strategies/paths in stochastic time-dependent networks, Nielsen, Pretolani, and Andersen [23]. Furthermore, in Nielsen [20], the algorithm is used as a sub algorithm to solve bicriterion problems in stochastic time-dependent networks.

By replacing departure times with states, and travel time with costs in the hypergraph model for stochastic time-dependent networks, it is apparent that hypergraphs also can be used to model MDPs. However, to the authors' knowledge no one has considered this way of modelling MDPs. Moreover, by modelling MDPs by hypergraphs we may find the K best policies by adapting existing K minimum weight hyperpath algorithms.

In this paper we consider finite-horizon Markov decision processes with finite state and action space and present an algorithm for finding the K best policies. That is, we are interested in ranking the first K policies in non-decreasing order using a certain optimality criterion. The algorithm uses a state-expanded directed hypergraph representing the finite-horizon Markov decision process. It is shown that the problem of finding the optimal policy can be formulated as a minimum weight hyperpath problem and be solved in linear time, with respect to the size of the input data[2] representing the MDP, using different optimality criterion.

Possible applications of the algorithm could involve presenting a set of near optimal policies for the decision maker from which he may pick the best policy fitting other preferences not contained in the Markov decision model. The algorithm can also be used for ranking of policies until a policy satisfying a hard constraint for the MDP is found. Note the constraints could be of different nature: constraints specifying that different types of costs must be below some given bounds or constraints imposed on the actual sample path, e.g. the policy must satisfy that action “maintain” must by used only once. Finally, an algorithm for finding the K best policies can be used to solve bicriterion problems in MDPs (using the two-phase approach). Here for instance the goal may be to minimize two objectives e.g. the expected total cost and the expected total risk. The latter perspective has served as the main motivation for the study, because a bicriterion optimization algorithm was needed in a forest management project with emphasis on risk.

The paper is organized as follows. Finite-horizon MDPs are introduced in Section 2 where also a short introduction to directed hypergraphs is given. In Section 3 a hypergraph model for MDPs is given together with results on how the best policy may be found. The algorithm for finding the K best policies is presented in Section 4. Conclusions are drawn in Section 5.

[2] See Section 2.1 for a formal definition of input data.

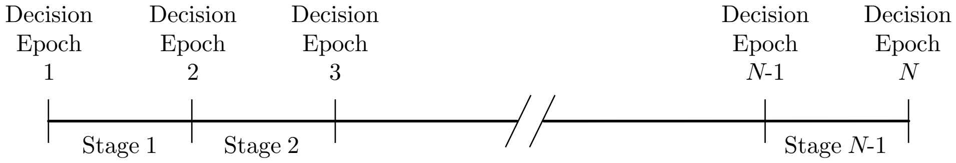


Figure 1: Decision epochs and stages.

2 Preliminaries

In this section preliminaries on finite-horizon Markov decision processes and directed hypergraphs are given.

2.1 Finite-horizon Markov decision processes

Markov decision processes are models for sequential decision making when outcomes are uncertain.

In this paper we consider a finite-horizon Markov decision process with $\{1, ..., N\}$ decisions and $N-1$ *stages* (see Figure 1). That is, decision number n is made at the beginning of stage n which corresponds to the time interval from decision number n to decision number $n+1$ (not including this time point).

At stage n the system occupies a *state*. We denote the finite set of system states S_n. Given the decision maker observes state $s \in S_n$ at stage n, he may choose an *action* a from the set of finite allowable actions $A_{s,n}$ generating *cost* $c_n(s, a)$ (a reward if negative). Moreover, we let $p_n(\cdot \mid s, a)$ denote the *probability distribution* or *transition probabilities* of obtaining states $s' \in S_{n+1}$ at stage $n+1$.

Since no decision is made at the end of stage $N-1$, the cost at this point of time is a function of the state $s \in S_N$ denoted $c_N(s, a_N)$ which is often referred to as the *salvage cost* or *scrap cost*. Here a_N denotes a deterministic (dummy) action.

A *deterministic Markovian decision rule* at stage n is a function $\mathfrak{d}_n : S_n \to A_{s,n}$ which specifies the action choice given state s at stage n. It is called deterministic because it chooses an action with certainty and Markovian (memoryless) since it depends only on the current system state. We let D_n denote the set of possible deterministic Markovian decision rules at stage n. D_n are a subset of more general rules where the action may depend on the past history of the system and actions may not be chosen with certainty but rather according to a probability distribution.

A *policy* or *strategy* specifies the decision rules to be used at all stages and provides the decision maker with a plan of which action to take given stage and state. That is, a policy δ is a sequence of decision rules, $\delta = (\mathfrak{d}_1, ..., \mathfrak{d}_N)$ with $\mathfrak{d}_n \in D_n$ for $n = 1, ..., N$. We restrict ourselves to ranking policies δ belonging to the set Δ of deterministic Markov policies (if randomized policies were included, the set of policies would not be countable). In some problems, the decision maker only focuses on this subset of policies, e.g because randomized policies are hard to manage in practice or restrictions in management strategy. Moreover, if the states at a given time instance corresponds to different physical locations implementation of policies having a single action at each location may only be acceptable.

In this paper we only consider one criterion of optimality, namely the expected total cost criterion. However, as pointed out in Section 3.1 the results can easily be extended to other criteria. Let X_n denote the state of the system at stage n, i.e. X_n is a random variable taking values in S_n. Then the expected total cost given policy δ and starting state s at stage 1 is

$$ETC^{\delta}\left(s\right)=\mathbb{E}_{s}^{\delta}\left(\sum_{n=1}^{N}c_{n}\left(X_{n},\mathfrak{d}_{n}\left(X_{n}\right)\right)\right) \tag{1}$$

In (1) we assume that the decision maker wishes to choose a policy given an initial system state s. In this paper we alternatively consider the case where he might seek a policy prior to knowing the initial state. Let $p_0\left(s\right)$ denote the probability of starting in state $s \in S_1$. In this case he seeks a policy $\delta \in \Delta$ which minimizes

$$ETC^{\delta}=\sum_{s\in S_{1}}p_{0}\left(s\right)ETC^{\delta}\left(s\right) \tag{2}$$

This corresponds to defining a policy to $\delta = \left(\mathfrak{d}_0, \mathfrak{d}_1, ..., \mathfrak{d}_N\right)$ where $\mathfrak{d}_0$ is the decision rule corresponding to a deterministic dummy action a_0. That is, we define stage 0 with $S_0 = \{s_0\}$ where s_0 represents the system before the state of the system at stage 1 is known and let $c_0\left(s_0, a_0\right)$ denote the cost. Moreover, we set $p_0\left(s' \mid s_0, a_0\right) = p_0\left(s'\right)$.

Let $\left|p_n\left(\cdot \mid s, a\right)\right|$ be the number of positive elements in the probability distribution and denote by

$$M=\sum_{n=0,\ldots,N,\;s\in S_{n},\;a\in A_{s,n}}\left|p_{n}\left(\cdot\mid s,a\right)\right| \tag{3}$$

the total number of possible transitions. Note that the size of the input data representing the MDP is $O\left(M\right)$.

It is well known that there exist a deterministic Markovian policy δ which minimizes (1), see e.g. Puterman [27, chap. 4]. Let *the cost to go* $u_n^{\delta}\left(s\right)$ denote the expected total cost given policy δ at stage $n, ..., N$, i.e.

$$u_{n}^{\delta}\left(s\right)=\mathbb{E}_{s}^{\delta}\left(\sum_{i=n}^{N}c_{i}\left(X_{i},\mathfrak{d}_{i}\left(X_{n}\right)\right)\right)$$

Then $u_n^{\delta}\left(s\right)$ can be found using the recursive equations

$$u_{n}^{\delta}\left(s\right)=\begin{cases}c_{n}\left(s,a\right)+\sum_{s'\in S_{n+1}}p\left(s'\mid s,a\right)u_{n+1}^{\delta}\left(s'\right) & n<N\\ c_{N}\left(s,a_{N}\right) & n=N\end{cases} \tag{4}$$

That is, the optimal policy with minimal expected total cost for all stages n and states $s \in S_n$ can be found using the following *Bellman equations*, Bellman [6]:

$$u_{n}^{*}\left(s\right)=\begin{cases}\min_{a\in A_{s,n}}\left\{c_{n}\left(s,a\right)+\sum_{s'\in S_{n+1}}p\left(s'\mid s,a\right)u_{n+1}^{*}\left(s'\right)\right\} & n<N\\ c_{N}\left(s,a_{N}\right) & n=N\end{cases} \tag{5}$$

indicating that the optimal policy can be found by analyzing a sequence of simpler inductively defined single-stage problems. This is often referred to as value iteration or backward induction (dynamic programming). The value iteration procedure for finding $u_n^{*}\left(s\right)$ for $n = 0, ..., N$ and $s \in S_n$ is shown in Figure 2.

```
procedure ValueIte
    for ($s \in S_N$) do $u^*_N(s) := c_N(s, a_N)$;
    for ($n = N - 1$ to 0) do
        for ($s \in S_n$) do
            find $u^*_n(s)$ using (5) and set $\mathfrak{d}^*_n(s)$ equal
            to the corresponding optimal action;
        end for
    end for
end procedure
```

Figure 2: The value iteration procedure.

2.2 Directed hypergraphs

A *directed hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = (v_1, ..., v_{|\mathcal{V}|})$ is the set of *nodes*, and $\mathcal{E} = (e_1, ..., e_{|\mathcal{E}|})$ is the set of *hyperarcs*. A hyperarc $e \in \mathcal{E}$ is a pair $e = (T(e), h(e))$, where $T(e) \subset \mathcal{V}$ denotes the set of *tail* nodes and $h(e) \in \mathcal{V} \setminus T(e)$ denotes the *head* node. Note that a hyperarc has exactly one node in the head, and possibly several nodes in the tail. A hypergraph is shown in Figure 3.

The *cardinality* of a hyperarc e is the number of nodes it contains, i.e. $|e| = |T(e)| + 1$. We call e an *arc* if $|e| = 2$. The *size* of $\mathcal{H}$ is the sum of the cardinalities. Without loss of generality, we assume $size(\mathcal{H}) > |V|$. We denote by

$$FS(v) = \{e \in \mathcal{E} \mid v \in T(e)\}, \quad BS(v) = \{e \in \mathcal{E} \mid v = h(e)\}$$

the *forward star* and the *backward star* of node v, respectively.

A hypergraph $\tilde{\mathcal{H}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ is a *subhypergraph* of $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, if $\tilde{\mathcal{V}} \subseteq \mathcal{V}$ and $\tilde{\mathcal{E}} \subseteq \mathcal{E}$. A subhypergraph is *proper* if at least one of the inclusions is strict.

An *valid ordering* $V = \left(v_1, v_2, \ldots, v_{|\mathcal{V}|}\right)$ of $\mathcal{H}$ is a topological ordering of the nodes such that, for any $e \in \mathcal{E}$, if $h(e) = v_i$ and $v_j \in T(e)$ then $j < i$. Note that, in a valid ordering any node $v_j \in T(e)$ precedes node $h(e)$. A *o-t path* in $\mathcal{H}$ is a sequence

$$(o = v_1, e_1, v_2, e_2, ..., e_q, v_{q+1} = t)$$

where, for $i = 1, \ldots, q$, $v_i \in T(e_i)$ and $v_{i+1} = h(e_i)$. A node v is *connected* to node u if a *u-v* path exists in $\mathcal{H}$. A *cycle* is a *o-t* path, where $t \in T(e_1)$. This is in particular true if $t = o$. If $\mathcal{H}$ contains no cycles, it is *acyclic*. It is well-known that $\mathcal{H}$ is acyclic if and only if a valid ordering of the nodes in $\mathcal{H}$ is possible (Gallo et al. [10]).

2.2.1 Hyperpaths and hypertrees

Definition 1 A hyperpath $\pi_{ot} = (\mathcal{V}_\pi, \mathcal{E}_\pi)$ from *origin* o to *target* t, is a subhypergraph of $\mathcal{H}$ satisfying that, if $t = o$, then $\mathcal{E}_\pi = \varnothing$; otherwise the $q \geq 1$ hyperarcs in $\mathcal{E}_\pi$ can be ordered in a sequence $(e_1, ..., e_q)$ such that

1. $t = h\left(e_q\right)$.

2. $T\left(e_i\right) \subseteq \{o\} \cup \left\{h\left(e_1\right), ..., h\left(e_{i-1}\right)\right\}, \quad \forall e_i \in \mathcal{E}_\pi$.

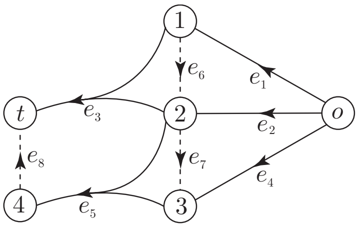


Figure 3: A hypergraph $\mathcal{H}$.

3. No proper subhypergraph of π_{ot} is an o-t hyperpath.

A node t is *hyperconnected* to o in $\mathcal{H}$ if there exists a hyperpath π_{ot} in $\mathcal{H}$. Note that condition 2 above implies that a valid ordering of π_{ot} is $(o, h(e_1), ..., h(e_q))$. That is, a hyperpath is *acyclic.* Furthermore, condition 3 implies that, for each $u \in \mathcal{V}_\pi \setminus \{o\}$, there exists a unique hyperarc $e \in \mathcal{E}_\pi$, such that $h(e) = u$ and hence for each node $u \in \mathcal{V}_\pi$ there is a unique subhyperpath π_{ou} contained in π_{ot}. We denote hyperarc e as the *predecessor* of u in π_{ot}. The definition of a hyperpath can be extended to hypertrees.

Definition 2 A directed hypertree of $\mathcal{H}$ with *root o* is an acyclic subhypergraph $\mathcal{T}_o = (\{o\} \cup \mathcal{N}, \mathcal{E}_\mathcal{T})$ with $o \notin \mathcal{N}$ satisfying

$$BS_\mathcal{T}(o) = \emptyset, \quad |BS_\mathcal{T}(v)| = 1, \quad \forall v \in \mathcal{N}$$

A directed hypertree $\mathcal{T}_o$ contains a unique o-u hyperpath for each node $u \in \mathcal{N}$ (see [10]). That is, $\mathcal{T}_o$ is the union of hyperpaths from o to all nodes in $\mathcal{N}$. Moreover, $\mathcal{T}_o$ can be described by a *predecessor function* $\mathfrak{g} : \mathcal{N} \to \mathcal{E}$; for each $u \in \mathcal{N}$, $\mathfrak{g}(u)$ is the unique hyperarc in $\mathcal{T}_o$ which has node u as the head. Note that any hyperpath is a hypertree, in particular, it can be defined by a predecessor function.

Example 1 A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is shown in Figure 3. $\mathcal{H}$ has a unique valid ordering, namely $V = (o, 1, 2, 3, 4, t)$. Below we give two hyperpaths in $\mathcal{H}$, namely a hyperpath from o to t and a hyperpath from o to 4.

$$\pi_{ot} = \big(\{o, 1, 2, t\}, \{e_1, e_2, e_3\}\big) \quad \pi_{o4} = \big(\{o, 2, 3, 4\}, \{e_2, e_4, e_5\}\big).$$

A hypertree $\mathcal{T}_o$ in $\mathcal{H}$ is shown with solid lines. It is the union of the two hyperpaths given above. Several valid orderings for $\mathcal{T}_o$ exist; one of them is $V = (o, 1, 2, t, 3, 4)$. ■

2.2.2 Weighting functions

Assume that each hyperarc e is assigned a real weight vector $w(e) = (w_1(e), ..., w_L(e))$. Given an o-t hyperpath π defined by predecessor function $\mathfrak{g}$, a weighting function W is a node function assigning real weights $W(u)$ to all nodes in π. The weight of hyperpath π is $W(t)$ (or $W(\pi)$). We shall restrict ourselves to *additive weighting functions* introduced by Gallo et al. [10], defined by the recursive equations:

$$W(v) = \begin{cases} 0 & v = o \\ \mathfrak{l}(w(\mathfrak{g}(v))) + \mathfrak{f}(\mathfrak{g}(v)) & v \in \mathcal{V}_\pi \setminus \{o\} \end{cases} \tag{6}$$

```
procedure $SHTacyclic(o, V, \mathcal{H})$
    $W(v_1) := 0$; for ($i = 2$ to $|\mathcal{V}|$) do $W(v_i) := \infty$;
    for ($i = 2$ to $|\mathcal{V}|$) do
        for ($e \in BS(v_i)$) do
            if ($W(v_i) > \mathfrak{l}(w(e)) + \mathfrak{f}(e)$) then
                $W(v_i) := \mathfrak{l}(w(e)) + \mathfrak{f}(e)$; $\mathfrak{g}(v_i) := e$;
        end for
    end for
end procedure
```

Figure 4: A procedure for finding the minimum weight hypertree in an acyclic hypergraph.

Here $\mathfrak{l}(\cdot)$ denote a non-decreasing function of $w(e)$ and $\mathfrak{f}(\cdot)$ a non-decreasing function of the weights in the nodes of $T(e)$. Furthermore, let $m_e(v)$ denote a nonnegative *multiplier* defined for each hyperarc e and node $v \in T(e)$. A particular case of (6) is the *value function* which has been studied in detail (see e.g. Gallo et al. [10], Jeroslow, Martin, Rardin, and Wang [13]) obtained by setting

$$W(v) = \begin{cases} 0 & v = o \\ w(\mathfrak{g}(v)) + \sum_{v \in T(\mathfrak{g}(v))} m_{\mathfrak{g}(v)}(v)\, W(v) & v \in \mathcal{V}_\pi \setminus \{o\} \end{cases} \tag{7}$$

2.2.3 Minimum weight hyperpaths

The *minimum weight hyperpath problem* or shortest hyperpath problem can be viewed as a natural generalization of the shortest path problem and consists in finding the minimum weight hyperpaths from a source o to all nodes in $\mathcal{H}$ hyperconnected to o. The result is a *minimum weight hypertree* containing minimum weight hyperpaths to all nodes hyperconnected to o.

If $\mathcal{H}$ is acyclic and the weighting function is additive a fast polynomial algorithm exist (see [10]). The procedure is shown in Figure 4 and needs a valid ordering $V = \left(o = v_1, v_2, \ldots, v_{|\mathcal{V}|}\right)$ of $\mathcal{H}$. Since each hyperarc is examined once, the procedure runs in $O(size(\mathcal{H}))$ time.

Example 1 (continued) Assume that each hyperarc e is assigned a scalar weight as shown in Figure 5 and multipliers $m_e(v) = 1/|T(e)|$, $v \in T(e)$. Then the weight of π_{ot} is 5 and the weight of π_{04} is 8 when the value weighting function (7) is used. Both hyperpaths are minimum weight hyperpaths to their corresponding target and the union of the two hyperpaths is a minimum weight hypertree. ■

3 A hypergraph model for finite-horizon Markov decision processes

Consider a finite-horizon Markov decision process with finite state and action spaces.

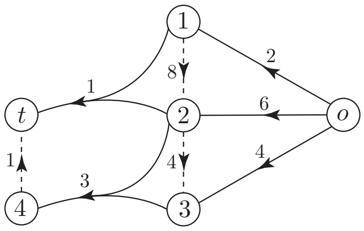


Figure 5: A hypergraph $\mathcal{H}$ with scalar weights.

(n,s)	$(1,1)$	$(1,2)$	$(2,1)$	$(2,2)$	$(3,1)$	$(3,2)$
$c_n\,(s,a)$	-70	-50	-70	-50	-70	-50
s'	$\{1,2\}$	$\{2,3\}$	$\{1,2\}$	$\{2,3\}$	$\{1,2\}$	$\{2,3\}$
$p_n\,(\cdot \mid s,a)$	$\left\{\frac{6}{10},\frac{4}{10}\right\}$	$\left\{\frac{6}{10},\frac{4}{10}\right\}$	$\left\{\frac{5}{10},\frac{5}{10}\right\}$	$\left\{\frac{5}{10},\frac{5}{10}\right\}$	$\left\{\frac{2}{10},\frac{8}{10}\right\}$	$\left\{\frac{2}{10},\frac{8}{10}\right\}$

Table 1: Input data for the problem Example 2 given action *nmt*.

Definition 3 Let the *state-expanded hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be obtained by defining the node and hyperarc set as follows

$$\mathcal{V} = \{v_{s,n} \mid n = 0, ..., N, s \in S_n\} \cup \{v_{N+1}\}$$

$$\mathcal{E} = \{e_{a,s,n} \mid n = 0, ..., N-1, s \in S_n, a \in A_{s,n}\} \cup \{e_{s,N} \mid s \in S_N\}$$

with

$$e_{a,s,n} = \left(\left\{v_{s',n+1} \mid s' \in S_{n+1},\, p_n\left(s' \mid s,a\right) > 0\right\}, v_{s,n}\right), \quad e_{s,N} = \left(\{v_{N+1}\}, v_{s,N}\right)$$

The following example illustrates how the state-expanded hypergraph is created.

Example 2 We consider a simple machine replacement problem. The state of the machine may be: good, average, and not working. Given the machine's state we may maintain the machine. In this case the machine's state will be good at the next decision epoch. Otherwise, the machine's state will not be better at next decision epoch. The machine is always replaced after 4 decision epochs. Furthermore, if the machine is not working then the machine may be replaced before decision epoch 4. Finally, when the machine is bought it may be either in state good or average.

The problem of when to replace the machine can be modelled using a Markov decision process with $N = 4$ decision epochs. We use system states *good* (1), *average* (2), and *not working* (3) together with actions *buy*, *maintain* (*mt*), *no maintenance* (*nmt*), and *replace* (*rep*). The system state sets S_n and action sets $A_{s,n}$ becomes

$$S_n = \begin{cases} \{s_0\} & n = 0 \\ \{1,2\} & n = 1 \\ \{1,2,3\} & n = 2,3,4 \end{cases}, \quad A_{s,n} = \begin{cases} \{\mathit{buy}\} & n = 0,\ s = s_0 \\ \{\mathit{mt}, \mathit{nmt}\} & n = 1,2,3,\ s = 1,2 \\ \{\mathit{mt}, \mathit{rep}\} & n = 2,3,\ s = 3 \\ \{\mathit{rep}\} & n = 4,\ s = 1,2,3 \end{cases}$$

The state set S_0 contains a single dummy state s_0 representing the machine before knowing its initial state and $A_{s_0,0}$ containing the deterministic action *buy*. Moreover, $A_{s,4}$, $s \in S_4$ contains the deterministic action *rep*.

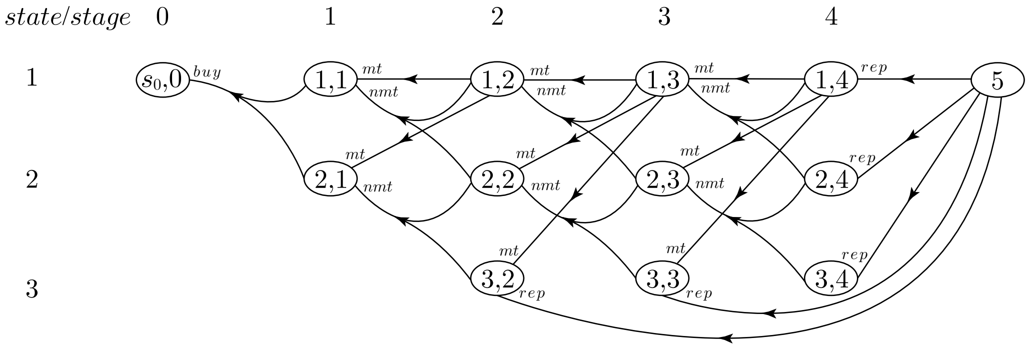


Figure 6: The state-expanded hypergraph.

The cost of buying the machine is 100 with $p_0(1) = 0.7$ and $p_0(2) = 0.3$. The reward (scrap value) of replacing a machine is 30, 10, and 5 in state 1, 2, and 3, respectively. The reward of the machine given action *mt* becomes 55, 40, and 30 given state 1, 2, and 3, respectively. Moreover, the system enters state 1 with probability 1 at the next stage. Finally, Table 1 shows the cost, transition states and probabilities given action *nmt*.

The state-expanded hypergraph $\mathcal{H}$ is shown in Figure 6 with the subscript of node $v_{s,n}$ shown in each node and action a corresponding to the hyperarc $e_{a,s,n}$ shown beside it. $\mathcal{H}$ contains a hyperarc $e_{a,s,n}$ for each possible action a given stage n and $s \in S_n$ and a node $v_{s,n} \in \mathcal{V}$ for all stages n and states $s \in S_n$. The head node of a hyperarc corresponds to the state of the system before action a is taken at the tail nodes to the possible system states after action a is taken. Furthermore, $\mathcal{H}$ contains a dummy node v_{N+1} which may be considered as the final system state representing the system after the machine has been replaced. Note this is often modelled using a dummy state *replaced* at each stage where the system stays in this state if it first enters it. However, this is avoided in the state-expanded hypergraph. Moreover, $\mathcal{H}$ contains arcs $e_{s,N}$, with tail node v_{N+1}, corresponding to the deterministic action *rep*. Finally, note that the direction of the hyperarcs are backward in time. ■

It is obvious that the head node of a hyperarc in the state-expanded hypergraph always corresponds to an earlier stage than any of its tail nodes (refer to Figure 6 for an illustration). Thus, no hyperarcs exists between nodes corresponding to the same stage. Hence the following property holds.

Property 1 *The state-expanded hypergraph $\mathcal{H}$ is acyclic. A valid ordering V is given by, first starting with node v_{N+1}, and next ordering the nodes $v_{s,n}$ in decreasing order for $n = 0, ..., N$. The nodes $v_{s,n}$, $s \in S_n$ for given n may be ordered arbitrarily in V.*

Moreover, since the size of the input data of the MDP is $O(M)$, the following is easily realized.

Property 2 *The state-expanded hypergraph can be built in $O(M)$ time and* size$(\mathcal{H}) = O(M)$.

Observe that there is a one to one correspondence between policies and predecessor functions $\mathfrak{g}$ on $\mathcal{H}$. Indeed, choosing $\mathfrak{g}(v_{s,n}) = e_{a,s,n}$ is equivalent to choosing $\mathfrak{d}_n(s) = a$. Moreover, $\mathfrak{g}(v_{s,N}) = e_{s,N}$ is the only possible predecessor for node $v_{s,N}$ indicating that only a deterministic dummy action a_N is possible at stage N. The same holds for node $v_{s_0,0}$.

Since a predecessor function $\mathfrak{g} : \mathcal{V}\backslash \{v_{N+1}\} \rightarrow \mathcal{E}$ according to Definition 2 define a hypertree with root v_{N+1} we have the following lemma.

Lemma 1 *Consider a finite-horizon MDP and its corresponding state-expanded hypergraph $\mathcal{H}$. Then the following holds*

1. *A hypertree $\mathcal{T}_{v_{N+1}} = (\{v_{N+1}\} \cup \mathcal{V}, \mathcal{E}_{\mathcal{T}})$ defined by predecessor function $\mathfrak{g}$ defines a policy δ.*

2. *A policy $\delta = (\mathfrak{d}_0, ..., \mathfrak{d}_N)$ defines a unique hypertree $\mathcal{T}_{v_{N+1}}$ in $\mathcal{H}$.*

Assign weights to the hyperarcs of $\mathcal{H}$ as follows

$$w_1 (e) = \begin{cases} c_n (s, a) & e = e_{a,s,n} \\ c_N (s, a_N) & e = e_{s,N} \end{cases} \tag{8}$$

Moreover, for each hyperarc e assign multipliers

$$m_e(v) = \begin{cases} p_n (s' \mid s, a) & e = e_{a,s,n},\ v = v_{s',n+1} \in T (e) \\ 1 & e = e_{s,N},\ v = v_{N+1} \end{cases} \tag{9}$$

Theorem 1 *Consider a policy δ defined by hypertree $\mathcal{T}_{v_{N+1}} = (\{o\} \cup \mathcal{V}, \mathcal{E}_{\mathcal{T}})$. Then the expected total cost $u^{\delta}_n (s)$, defined in* (4)*, is equal to the weight $W (v_{s,n})$ found using the value weighting function with weights* (8) *and multipliers* (9)*.*

Proof Consider the recursive definition of the value weighting function (7), applied to the nodes of the hypertree. For node $v_{s,N}$, $s \in S_N$, we have that

$$W (v_{s,N}) = c_N (s, a_N) = u^{\delta}_N (s_N)$$

For node $v_{s,n}$, $n = 0, ..., N - 1$ with predecessor hyperarc $e_{a,s,n}$, we have that

$$\begin{aligned} W (v_{s,n}) &= w (e_{a,s,n}) + \sum_{v \in T(e)} m_e (v) W (v) \\ &= c_n (s, a) + \sum_{s' \in S_{n+1}} p (s' \mid s, a) u^{\delta}_{n+1} (s) \end{aligned}$$

Since the recursive definitions of W and u^{δ}_n are identical, $W (v_{s,n}) = u^{\delta}_n (s)$. ■

Theorem 1 implies that an optimal policy δ with $u^{\delta}_n (s) = u^*_n (s)$ can be found by finding an optimal predecessor function in $\mathcal{H}$. Moreover, by using procedure *SHTacyclic*, shown in Figure 4, this can be done in $O (size (\mathcal{H}))$ time, i.e. linear in the size of the input data representing the MDP.

Corollary 1 *The problem of finding an optimal policy δ with $u^{\delta}_n (s) = u^*_n (s)$, for all $n = 0, ..., N$, $s \in S_n$ can be formulated as a minimum weight hyperpath problem on $\mathcal{H}$ and solved in $O (M)$ time.*

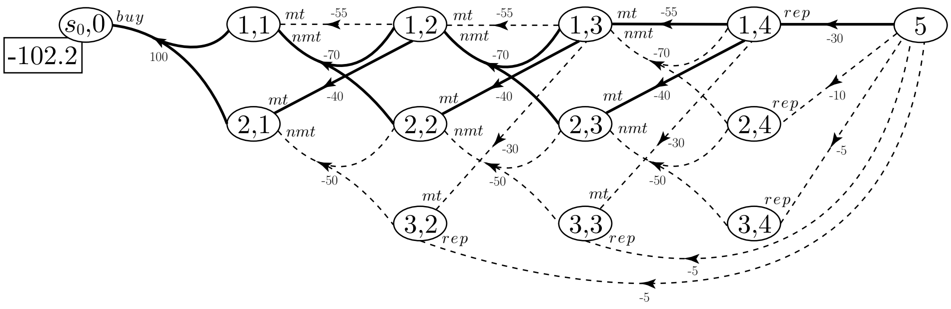


Figure 7: The optimal policy.

Note that the worst case complexity of procedure *ValueIte* in Figure 2 and procedure *SHTacyclic* in Figure 4 are the same. Procedure *SHTacyclic* may be considered as a variant of procedure *ValueIte* using another underlying data structure, namely, a directed hypergraph.

Example 2 (continued) Assume that weights (8) and multipliers (9) are assigned to the hyperarcs of the state-expanded hypergraph. Then the optimal policy can be found using procedure *SHTacyclic* shown in Figure 4. The hyperpath corresponding to the optimal policy is shown in bold in Figure 7. The expected total reward is 102.2. Note each time the machine reaches the *average* state it is maintained. ■

3.1 Other criteria of optimality

Besides the minimization of the expected total cost several other criteria for selecting an optimal policy can be taken into account. For example, we may assign a discounting rate $\lambda_n(s,a)$ of taking action a in state s at stage n. Let the weight of hyperarc e be defined as follows

$$w(e) = (w_1(e), w_2(e)) = \begin{cases} (c_n(s,a), \lambda_n(s,a)) & e = e_{a,s,n} \\ (c_N(s,a_N), 0) & e = e_{s,N} \end{cases} \tag{10}$$

Then the expected total discounted reward can be found using weights (10) and multipliers (9) and the weighting function (6) with

$$\mathfrak{l}(w(e)) = c_n(s,a), \quad \mathfrak{f}(e) = \lambda_n(s,a) \sum_{v \in T(e)} m_e(v) W(v)$$

The proof is similar to the one for Theorem 1. Other criteria such as the expected total reward per unit time may be modelled similarly.

Another possibility is to consider worst cases (maximum possible cost), rather than the average behaviour. This situation can be modelled, with a proper choice of hyperarc weights and weighting function. For instance, the policy minimizing the maximum possible total cost can be found using weighting function (6) with

$$\mathfrak{l}(w(e)) = w_1(e) = c_n(s,a), \quad \mathfrak{f}(e) = \max_{v \in T(e)} \{W(v)\}$$

This weighting function is also known as the *distance* in the literature (see e.g. Gallo et al. [10], Nielsen, Andersen, and Pretolani [22]).

4 Finding the K best policies

Consider the state-expanded hypergraph $\mathcal{H}$ of a finite-horizon MDP. We consider the problem of finding the K best policies. That is, ranking the first K policies in non-decreasing order using criterion (2). According to Theorem 1 a policy δ corresponds to a hypertree $\mathcal{T}$ with root v_{N+1}. Moreover, the expected total cost prior to knowing the initial state is equal the weight of the hyperpath $\pi \subseteq \mathcal{T}$ with origin v_{N+1} and target $v_{s_0,0}$. Hence finding the K best policies using criterion (2) corresponds to finding the K minimum weight hyperpaths from origin v_{N+1} to target $v_{s_0,0}$ in the state-expanded hypergraph using the value weighting function with weights (8) and multipliers (9).

Efficient algorithms for finding the K minimum weight hyperpaths were developed by Nielsen et al. [21]. These algorithms are based on an implicit enumeration method, where the set of hyperpaths is partitioned into smaller subsets by recursively applying a *branching operation*. In the following we give a short description of the algorithm.

Let Π denote the set of hyperpaths in $\mathcal{H}$ with origin $o = v_{N+1}$ and target $t = v_{s_0,0}$. Consider the minimum weight hyperpath π defined by predecessor function $\mathfrak{g}$ and with valid ordering

$$V_\pi = (o, u_1, ...u_q = t)$$

Given the minimum weight hyperpath π of Π and valid ordering V_π, the set $\Pi \setminus \{\pi\}$ can now be partitioned into q disjoint subsets Π^i, $1 \leq i \leq q$ using the following branching operation (for a formal proof see Nielsen [20]).

1. Hyperpaths in Π^q do not contain hyperarc $\mathfrak{g}(u_q)$, that is $\mathfrak{g}(t)$.
2. For $1 \leq i < q$, hyperpaths in Π^i contain hyperarcs $\mathfrak{g}(u_j)$, $i+1 \leq j \leq q$, and do not contain hyperarc $\mathfrak{g}(u_i)$.

It is evident that by taking the minimum weight hyperpath in $\cup_{i=1,\dots,q}\Pi^i$ we find the second minimum weight hyperpath. Furthermore, the branching operation can be applied to the second minimum weight hyperpath recursively.

Let δ denote a optimal policy. In terms of MDPs the above branching operation corresponds to partition the set $\Delta \setminus \{\delta\}$ of deterministic Markov policies into q disjoint subsets from which the second best policy can be found.

The branching operation partitions the set Π in a way that simplifies finding the minimum weight hyperpath in each subset. Indeed, finding the minimum weight hyperpath $\pi^i \in \Pi^i$, $i = 1, ..., q$, reduces to solving a minimum weight hyperpath problem on the subhypergraph $\mathcal{H}^i$ obtained from $\mathcal{H}$ as follows:

1. For each node u_j, $i+1 \leq j \leq q$, remove each hyperarc in $BS(u_j)$ except $\mathfrak{g}(u_j)$.
2. Remove hyperarc $\mathfrak{g}(u_i)$ from $BS(u_i)$.

We say that hyperarc $\mathfrak{g}(u_j)$, $i+1 \leq j \leq q$ is *fixed*, since all other hyperarcs have been removed from the backward star of u_j. Note that subhypergraph $\mathcal{H}^i$ is the state-expanded hypergraph for the MDP where the actions corresponding to hyperarc $\mathfrak{g}(u_{i+1}), ..., \mathfrak{g}(u_q)$ must be taken and the action corresponding to hyperarc $\mathfrak{g}(u_i)$ cannot be taken.

Given subhypergraph $\mathcal{H}^i$ the following are equivalent for $i = 1, ..., q$.

```
procedure $K\text{-}BP(\mathcal{H}, o, t, K)$
    $SHTacyclic(o, \mathcal{H})$;
    if $(W(t) < \infty)$ then $insert((\pi, \mathcal{H}))$;
    else STOP (there is no $o$-$t$ hyperpath);
    for $(k := 1 \text{ to } K)$ do
        $(\tilde{\pi}, \tilde{\mathcal{H}}) := delMin()$;
        if $((\tilde{\pi}, \tilde{\mathcal{H}}) = null)$ then STOP (there are no more $o$-$t$ hyperpaths);
        OUTPUT the $k$'th hyperpath $\tilde{\pi}$;
        $(o, u_1, ..., u_q) := findV(\tilde{\pi})$;
        for $(i := q \text{ to } 1)$ do
            $\tilde{\mathcal{H}}^i := findH(\tilde{\mathcal{H}}, u_i)$;
            $\tilde{\pi}^i := findPi(\tilde{\mathcal{H}}^i)$;
            $insert((\tilde{\pi}^i, \tilde{\mathcal{H}}^i))$;
        end for
    end for
end procedure
```

Figure 8: Finding the K best policies.

1. $\pi \in \Pi^i$.
2. π is an o-t hyperpath in $\mathcal{H}^i$.

As a consequence, each set Π^i can be represented by its corresponding subhypergraph $\mathcal{H}^i$. Hence in order to find the K minimum weight hyperpaths, we implicitly have to maintain a candidate set of pairs $(\tilde{\pi}, \tilde{\mathcal{H}})$, where $\tilde{\pi}$ is a minimum weight hyperpath in $\tilde{\mathcal{H}}$. Assuming that the first k minimum weight hyperpaths $\pi_1, ..., \pi_k$ have been found, the candidate set represents a partition of $\Pi \setminus \{\pi_1, ..., \pi_k\}$. Hyperpath π_{k+1}, i.e. the $K + 1$'th best policy, is then found by picking and removing the pair representing the hyperpath with minimum weight in the candidate set. Then the branching operation is applied using hyperpath π_{k+1}, possibly obtaining new pairs that are added to the candidate set.

A compact version of the algorithm for finding the K minimum weight hyperpaths in an acyclic hypergraph, i.e. finding the K best policies is shown in Figure 8. The following subprocedures are used.

SHTacyclic$(o, \mathcal{H})$: Find the minimum weight hypertree of $\mathcal{H}$, i.e the best policy (represented by π) of the MDP corresponding to $\mathcal{H}$ (see Figure 4).

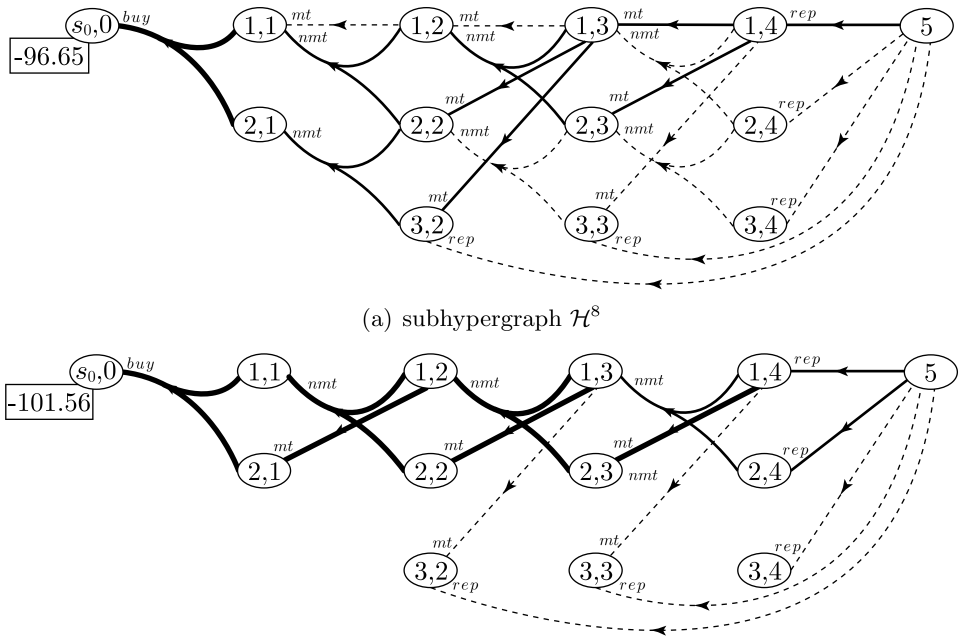
delMin$()$: Select, remove and return the pair $(\tilde{\pi}, \tilde{\mathcal{H}})$ with minimum hyperpath weight from the candidate set.

insert$((\tilde{\pi}, \tilde{\mathcal{H}}))$: Insert the pair $(\tilde{\pi}, \tilde{\mathcal{H}})$ into the candidate set.

findV$(\tilde{\pi})$: Return a valid ordering $V_{\tilde{\pi}} \subseteq V_{\mathcal{H}}$ of the nodes in $\tilde{\pi}$.

findH$(\mathcal{H}, u_i)$: Create and return the subhypergraph $\mathcal{H}^i$.

findPi$(\mathcal{H}^i)$: Find and return the minimum weight hyperpath of $\mathcal{H}^i$.



(a) subhypergraph $\mathcal{H}^8$

(b) subhypergraph $\mathcal{H}^3$

Figure 9: Subhypergraph $\mathcal{H}^8$ and $\mathcal{H}^3$.

The best policy, i.e. the minimum weight hyperpath π of $\mathcal{H}$ is found and inserted into the candidate set on line 2-4. If no hyperpath exists, i.e. the weight $W(t)$ of the hyperpath is infinity, then no optimal policy exists. Line 5-15 contains the main loop. In the k'th iteration the minimum weight pair $(\tilde{\pi}, \tilde{\mathcal{H}})$ is picked and we output the k'th policy. The branching operation is performed on line 10-14.

Different implementations of the procedures *findH* and *findPi* have been given in the literature. The algorithm from Nielsen et al. [24] find $\tilde{\pi}$ using procedure *SHTacyclic* each time a pair is selected form the candidate set. Moreover, in the branching operation, each hyperpath $\tilde{\pi}^i$ is found using procedure *SHTacyclic* on subhypergraph $\tilde{\mathcal{H}}^i$. The main drawback of this algorithm is that a minimum weight hyperpath problem must be solved for each subhypergraph generated during the branching operation. The number of minimum weight hyperpath problems to solve is therefore much larger than K.

In Nielsen et al. [21], the complexity of the algorithm is improved by using reoptimization techniques to avoid solving minimum weight hyperpath problems. Here procedure *SHTacyclic* is called only once, namely when the minimum weight hyperpath of $\mathcal{H}$ is found. Afterwards, hypergraph $\tilde{\mathcal{H}}$ and hyperpath $\tilde{\pi}$ is created implicitly by reusing information. Finally, the minimum weight hyperpath $\tilde{\pi}^i$ in subhypergraph $\tilde{\mathcal{H}}^i$ is found using simple calculations on the hyperarcs in the backward star of the node u_i. By using reoptimization the complexity can be improved to $O(size(H)K)$. Hence we have the following theorem.

Theorem 2 *Procedure K-BP finds the K best policies in worst time complexity $O(MK)$.*

Example 2 (continued) Due to outside safety regulations assume that the probability of a machine being maintained more than once during its lifetime must be zero. That is, the

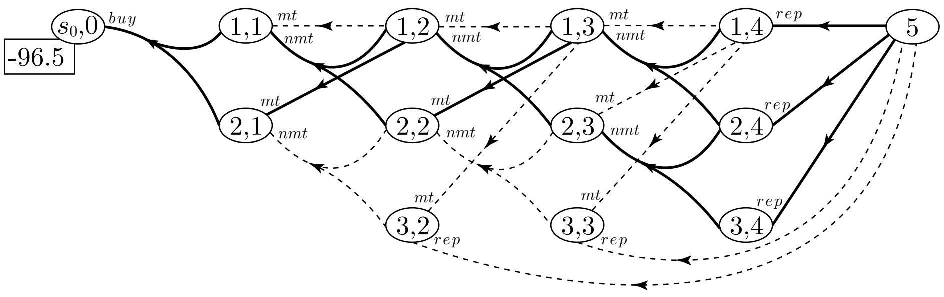


Figure 10: The optimal policy under the constraint ($k = 10$).

actual sample path of the policy must satisfy that action "maintain" is used at most once.

This constraint corresponds to the hyperpath with origin $o = v_{N+1}$ and target $t = v_{s_0,0}$ may only contain *o*-*t* paths with at most a single arc corresponding to action *mt*. Note that for a given policy this can be checked in $O(|\mathcal{V}|)$ time by visiting the nodes in the corresponding hyperpath using the valid ordering of the time-expanded hypergraph.

The constraint is not fulfilled for the optimal policy, i.e. the hyperpath π_1 in Figure 7. We use procedure *K-BP* to rank the policies until a policy satisfying the constraint is found. A valid ordering of π_1 (defined by predecessor function $\mathfrak{g}$) is

$$V_{\pi_1} = (o, u_1, ..., u_9) = (o, v_{1,4}, v_{2,4}, v_{1,3}, v_{2,3}, v_{1,2}, v_{2,2}, v_{1,1}, v_{2,1}, v_{s_0,0} = t)$$

Hence using the branching operation on π_1 corresponds to creating 9 subhypergraphs $\mathcal{H}^9, ..., \mathcal{H}^1$. Subhypergraph $\mathcal{H}^9$ is created by removing $\mathfrak{g}\,(u_9)$, i.e. action *buy*. Since $\mathfrak{g}\,(u_9)$ is the only hyperarc in the backward star of node u_9, we have that no *o*-*t* hyperpath exists in $\mathcal{H}^9$, that is, no policy exists and it can be ignored. Subhypergraph $\mathcal{H}^8$, shown in Figure 9(a), is obtained by fixing $\mathfrak{g}\,(u_9)$ (shown in bold) and removing $\mathfrak{g}\,(u_8)$, i.e action *maintain* in state average at time instance one. The minimum weight hyperpath of $\mathcal{H}^8$ are shown with solid lines. The expected total reward of the policy is 96.65.

Similar we find $\mathcal{H}^7, ..., \mathcal{H}^1$ and their corresponding minimum weight hyperpath. Comparing the hyperpath weights of $\mathcal{H}^9, ..., \mathcal{H}^1$, the second best policy is the policy corresponding to the minimum weight hyperpath of $\mathcal{H}^3$, shown in Figure 9(b), with expected total reward 101.56.

Note that this policy does not satisfy the constraint and the branching operation is used on the second best policy. The branching operation is repeated recursively until the first policy which fulfill the constraint is found which is the $k = 10$'th policy with reward 96.5, shown in Figure 10. ■

5 Conclusions

In this paper we considered the problem of finding the K best policies in a finite-horizon Markov decision processes with finite state and action space. That is, ranking the first K policies in non-decreasing order using a certain optimality criterion. Finite-horizon MDPs have been considered for many years. However, the problem of finding the K best policies has not yet been solved. The results in this paper were motivated by recent results on stochastic time-dependent networks which have similarities to finite-horizon MDPs. The main contributions of this paper can be summarized as follows.

A finite-horizon MDP can be modelled using a state-expended directed hypergraph. Even though several hyperpath models have been proposed in the literature, no one have modelled finite-horizon MDPs using hypergraphs.

Hyperpaths in the state-expanded hypergraph are equivalent to a policy in the MDP. As a result the problem of finding the optimal policy can be formulated as a minimum weight hyperpath problem and be solved in linear time, with respect to the size of the input data of the MDP. Moreover, different optimality criterion can easily be modelled using different weights and weighting functions on the state-expanded hypergraph.

Since a policy corresponds to a hypertree $\mathcal{T}$ and the expected total cost prior, to knowing the initial state, is equal the weight of the v_{N+1}-$v_{s_0,0}$ hyperpath $\pi \subseteq \mathcal{T}$ we have that finding the K best policies using criterion (2) corresponds to finding the K minimum weight v_{N+1}-$v_{s_0,0}$ hyperpaths using the value weighting function with weights (8) and multipliers (9). Hence using recent efficient algorithms for finding the K shortest hyperpaths, Nielsen et al. [21, 24], we have that the K best policies can be found in $O\left(MK\right)$ time.

Possible applications are ranking policies until a policy satisfying a hard constraint for the MDP is found and solving bicriterion problems in MDPs. Here for instance, the objective may be to minimize both the expected total cost and the total risk.

Directions for further research include finding the K best deterministic Markov policies in a infinite-horizon MDP, modelling random policies using directed hypergraphs and hypergraph models for influence diagrams, see Nilsson [25].

Acknowledgements

The main part of this research was supported by a grant from SNS - the Nordic Forest Research Cooperation Committee. Initial work was done at the Danish Institute of Agricultural Sciences in the project *Farrowing and nursing sows, loose housing and improved animal welfare* funded by The Danish Directorate for Food, Fisheries and Agri Business.

We are grateful to scientist Erik Jørgensen for useful comments and suggestions for improving this paper.

References

[1] E. Altman. *Constrained Markov Decision Processes.* Chapman and HALL/CRC, 1999.

[2] E. Altman and A. Shwartz. Adaptive control of constrained Markov chains: Criteria and policies. *Annals of Operations Research*, 28:101–134, 1991.

[3] G. Ausiello, A. Datri, and D. Sacca. Minimal representation of directed hypergraphs. *SIAM Journal on Computing*, 15(2):418–431, 1986.

[4] G. Ausiello, P.G. Franciosa, and D. Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. *Lecture Notes in Computer Science*, 2202:312–328, 2001.

[5] Giorgio Ausiello, Giuseppe F. Italiano, and Umberto Nanni. Hypergraph traversal revisited: Cost measures and dynamic algorithms. In *Mathematical Foundations of Computer Science: 23rd International Symposium*, volume 1450 of *Lecture Notes in Computer Science*, pages 1–16, August 1998. URL `http://www.springerlink.com/index/1RE5B6E5DUGN6Y6N`.

[6] R. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[7] C. Berge. *Graphs and hypergraphs.* North-Holland, 1973.

[8] C. Derman and Jr. A.F. Veinott. Constrained Markov decision chains. *Management Science*, 19:389–390, 1972.

[9] C. Derman and M. Klein. Some remarks on finite horizon Markovian decision models. *Operations Research*, 13:272–278, 1965.

[10] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993.

[11] A. Hordijk and L.C.M. Kallenberg. Constrained undiscounted stochastic dynamic programming. *Mathematics of Operations Research*, 9:276–289, 1984.

[12] R.A. Howard. *Dynamic Programming and Markov processes.* Cambridge, Massachusetts: The M.I.T. Press, 1960.

[13] R.G. Jeroslow, K. Martin, R.L. Rardin, and J. Wang. Gainfree Leontief substitution flow problems. *Mathematical Programming*, 57:375–414, 1992.

[14] L.C.M. Kallenberg. Linear programming and finite Markovian control problems. *Mathematical Centre Tracts*, 148, 1983. Amsterdam.

[15] L.C.M. Kallenberg. Survey of linear programming for standard and nonstandard Markovian control problems, part I: Theory. *ZOR - Methods and models in Operations Research*, 40:1–42, 1994.

[16] A.R. Kristensen. A general software system for Markov decision processes in herd management applications. *Computers and Electronics in Agriculture*, 38(3):199–215, 2003.

[17] A.R. Kristensen. Hierarchic Markov processes and their applications in replacement models. *European Journal of Operational Research*, 35:207–215, 1988.

[18] A.R. Kristensen and E. Jørgensen. Multi-level hierarchic Markov processes as a framework for herd management support. *Annals of Operations Research*, 94(1):69–90, 2000.

[19] S. Nguyen and S. Pallottino. Hyperpaths and shortest hyperpaths. In *Combinatorial optimization (Como, 1986)*, volume 1403 of *Lecture Notes in Math*, pages 258–271. Springer, 1989.

[20] L.R. Nielsen. *Route Choice in Stochastic Time-Dependent Networks.* PhD thesis, Department of Operations Research, University of Aarhus, 2004.

[21] L.R. Nielsen, D. Pretolani, and K.A. Andersen. Finding the K shortest hyperpaths using reoptimization. *Accepted for publication in Operations Research Letters.* doi:10.1016/j.orl.2005.04.008.

[22] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Bicriterion shortest hyperpaths in random time-dependent networks. *IMA Journal of Management Mathematics*, 14(3):271–303, 2003.

[23] L.R. Nielsen, D. Pretolani, and K.A. Andersen. K shortest paths in stochastic time-dependent networks. Technical Report WP-L-2004-05, Department of Accounting, Finance and Logistics, Aarhus School of Business, 2004. URL `http://www.asb.dk/departments/afl/research/lrg/workingpapers/`. Submitted.

[24] L.R. Nielsen, K.A. Andersen, and D. Pretolani. Finding the K shortest hyperpaths. *Computers & Operations Research*, 32(6):1477–1497, 2005. doi:10.1016/j.cor.2003.11.014.

[25] D. Nilsson. *Finding Sets of Most Probable Configurations in Bayesian Networks and Best Strategies in Infludence Diagrams.* PhD thesis, Department of Mathematics, Aalborg University, 1998.

[26] D. Pretolani. A directed hypergraph model for random time-dependent shortest paths. *European Journal of Operational Research*, 123:315–324, 2000.

[27] M.L. Puterman. *Markov Decision Processes.* Wiley Series in Probability and Mathematical Statistics. Wiley-Interscience, 1994.

[28] G. Ramalingam and Thomas Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267–305, 1996.