Control and Cybernetics

vol. **41** (2012) No. 3

Solving Markov decision processes by d–graph algorithms[*]

by

Zoltán Kátai

Sapientia University,
Tîrgu Mureş, Romania

Abstract: Markov decision processes (MDPs) provide a mathematical model for sequential decision making (sMDP/dMDP: stochastic/deterministic MDP). We introduce the concept of generalized dMDP (g-dMDP) where each action may result in more than one next (parallel or clone) state. The common tools to represent dMDPs are digraphs, but these are inadequate for sMDPs and g-dMDPs. We introduce d-graphs as general tools to represent all the above mentioned processes (stationary versions). We also present a combined d-graph algorithm that implements dynamic programming strategies to find optimal policies for the finite/infinite horizon versions of these Markov processes. (The preliminary version of this paper was presented at the Conference MACRo 2011.)

Keywords: Markov decision processes, dynamic programming, graph representation, graph algorithms, optimization problems.

1. Introduction

A Markov decision process (MDP) is a mathematical tool for modeling sequential decision making under uncertainty. MDP models have gained recognition in numerous fields of science like optimal control, operations research, AI, economics, game theory, computer sciences, telecommunications, etc.

The term of MDP is historically related to the notions of “Dynamic programming” and “Markov chain”. The first book in dynamic programming (DP) was published by Richard Bellman in 1957 (Bellman, 1957). The book presents DP as a new numerical method for solving sequential decision problems. Three year later, in 1960, Ronald Howard published the book entitled *”Dynamic Programming and Markov Processes”* (Howard, 1960). This work combines the dynamic programming technique with the mathematically well-established notion of Markov chain. The term of MDP naturally resulted from this combination (see Kristensen, 1996). During the past fifty years MDPs have become useful tools for studying a wide range of optimization problems solved via dynamic programming.

[*]Submitted: May 2011; Accepted: September 2012

A DP problem can usually be formulated either as an optimal structure or an optimal control problem. By representing a DP problem as a graph we move it to a well developed area: graph theory. The most common methods use digraphs. This approach often leads to an optimal path problem. Since the ordinary digraph representation is not always sufficient, Lew introduced Bellman nets (specialized acyclic Petri nets) to model complex optimal structure DP problems (see Lew, 2002). These problems may have optimal solutions which are represented by K-ary trees (rather than paths). When motivating their choice for Petri Nets, Lew and Mauch (2004; 2007) explain why – because of the overlapping sub-problems – a tree or a parse-tree model would also be inappropriate. The hierarchic d-graph model introduced by Katai (2006) also solves both impediments mentioned by Lew and Mauch.

The DP problem solving process can be divided into two steps: (a) the functional equation of the problem is established (a recursive formula that implements the principle of optimality); (b) a computer program is elaborated that processes the recursive formula, usually in a bottom-up / backward way. The second step commonly means that specialized software is developed for every problem in particular. To save software development costs, authors like Lew and Katai proposed general software tools automatically solving optimal structure discrete DP problems. The solver software DP2PN2Solver uses Bellman nets as intermediate representation of the functional equation (Mauch, 2006; Lew and Mauch, 2007). The software tool proposed by Katai and Csiki (2009) builds up the d-graph representation of the problem.

The analysis presented by Kátai and Fülöp (2010) compares the two methods and software tools. The Bellman net model works only when the functional equation is formulated in such a way that it does not contain "circular definitions" (see Lew and Mauch, 2007). Since the functional equation describes the way the optimal solution of the current sub-problem *may* depend on the optimal solution of other sub-problems we should not exclude "structurally circular definitions" (in Section 2 we present an example). Katai (2010) extended the concept of d-graph to generalized d-graph that can handle DP problems with "cyclic functional equations" too. This quality of d-graphs is also essential when we use them to represent MDPs. Lew and Mauch (2007) mentioned the possibility to extend their model to iterative problems associated with MDPs among future research plans.

In this paper we extend the d-graph representation method to optimal control problems, more exactly to MDPs. We introduce the concept of generalized deterministic MDP to extend the model to cases where each action may result in more than one next (parallel or clone) state. We present a combined d-graph algorithm that implements dynamic programming strategies to find optimal policies for the finite/infinite horizon versions of the MDPs. We plan to develop a software tool that automatically solves MDPs (the input being the functional equation).

2. Markov decision processes

An MDP includes a finite set of states (S), one of the states being identified as the initial state of the system ($s_0 \in S$). In each time unit or stage, the controller of the MDP has to select an action from the subset of currently enabled actions ($A(s), s \in S$). An MDP also includes an immediate cost (or reward) and a probability distribution over the next state of the process associated with each action ($C(s,a) \in R, P(s,a) \in$[0,1]$, s \in S, a \in A(s)$). In a finite/infinite horizon MDPs, the controller has to guide the process for a finite/infinite number of steps. Our goal is to optimize the overall cost/reward associated with the performed action-sequence. Whereas for the finite horizon version of the problem the sum of immediate costs of the action-sequence is a proper overall cost/reward function, for infinite horizon problems this function is often chosen as the limit of the average cost/reward, or the total discounted cost/reward (see Madani, Thorup and Zwick, 2009).

A policy (or strategy) for an MDP is a mapping that defines the action the controller has to perform in each state ($\pi : S \rightarrow A$). The decision concerning the current action may depend on the current state of the process and possibly on history. A policy is called positional if it is pure and history independent (markovian deterministic policy), see Gimbert (2008). A policy that minimizes/maximizes the overall cost/reward is called optimal (π^*). One of the main results for MDPs is that an infinite horizon MDP always has exactly one optimal positional policy which is optimal for every starting-state (Puterman, 1994). Solving an MDP means finding such an optimal policy and the optimal overall cost for each starting-state (see Madani, Thorup and Zwick, 2009).

An MDP is said to be *deterministic* (dMDP) if each action uniquely defines (with probability 1) the next state of the process. (For *stochastic* MDP we use the acronym sMDP further on) We extend the definition of dMDP to *generalized* dMDP (g-dMDP) where each action may result (with probability 1) in more than one next state. We call these parallel states *clone-states.* To illustrate the necessity of this extension we present the following optimal control problem.

2.1. An illustrating example

(1^{st} version) A discrete bi-dimensional world can be characterized by a finite number of world states $\{(i,j) \ |i$=1..N, j =1..$M\}$. This world can intuitively be imagined as a bi-dimensional array s(1..N,1..M). An agent enters array s at cell (1,1) and leaves it at cell (N,M). At each time step the agent has to select among given valid actions (for example, to move to (N)orth/(E)ast/(S)outh/(W)est). Each action has an immediate cost: in each cell that the agent entered a tax has to be paid ($s(i,j)$ represents the tax attached to cell (i,j)). We are interested in the optimal (minimal overall cost) way of getting through the world. If we denote the optimal cost from cell (i,j) to cell (N,M) with $c(i,j)$ then the optimal overall cost will be represented by the value c(1,1). This problem, in this original version, can properly be modeled by a dMDP. Fig. 1 shows a sample world where there are bidirectional doors between all neighboring cells (see also Fig. 4).

In this case the most natural form of the functional equation is the following:

$$c\left(N,M\right)=s\left(N,M\right)$$

Otherwise:

$$c\left(i,j\right)=s\left(i,j\right)+min(c\left(i-1,j\right),c\left(i,j+1\right),c\left(i+1,j\right),c\left(i,j-1\right))$$

(assuming that the corresponding neighbor-cells exist).

Note, for example, that (on the basis of the structural dependences built-in the above recursive formula) the value c(3,2) *may* depend on value c(3,3), and conversely, value c(3,3) *may* depend on value c(3,2):

$$c\left(3,2\right)=s\left(3,2\right)+min(c\left(2,2\right),c\left(3,3\right),c\left(4,3\right),c\left(3,1\right))$$

$$c\left(3,3\right)=s\left(3,3\right)+min(c\left(2,3\right),c\left(3,4\right),c\left(4,3\right),c\left(3,2\right)).$$

At the end of the optimization process we have
c (3,3)= s (3,3)+c (3,2).
(Value c (3,3) depends on value c (3,2).)

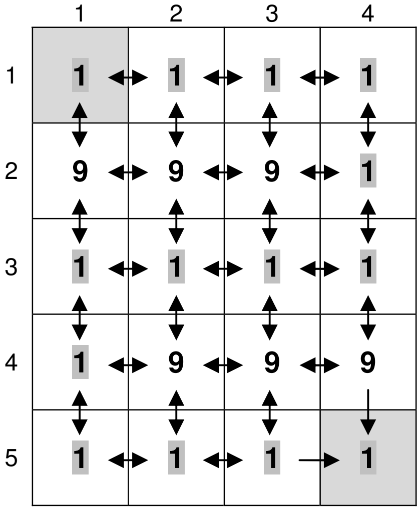


Figure 1. Bi-dimensional world (N =5, M =4). Starting cell: (1,1). Goal cell: (5,4). Between all neighbor cells there are bidirectional doors. The optimal path from the starting cell to the goal cell is marked

(2^{nd} version) Let us now consider a second version of the problem: The agent may create (or clone) further copies (instances) of himself. Accordingly, an instance that entered a cell may go on from there to more than one direction in parallel (cloning as many further copies as necessary). We consider the original instance that enters the world as being a 100% taxable agent ("100% – instance"). If the current cell is entered by an "x% – instance" and left by n instances (the entered one and his newly created clones), then the leaving instances are considered "(x/n)% – instances". An "x% – instance" has to pay only x% of the tax attached to the cell it has entered. Since a cell may have maximum of 4 "out-doors" there are maximum 2^4-1 possibilities for the entered instance to go on. For each cell a hex-string codes the corresponding valid actions. Fig. 2 shows the 15 possibilities to go on from the current cell (see also Fig. 7). We are interested in the optimal way the clone population can traverse the world. The most appropriate model for this second version of the problem is a g-dMDP.

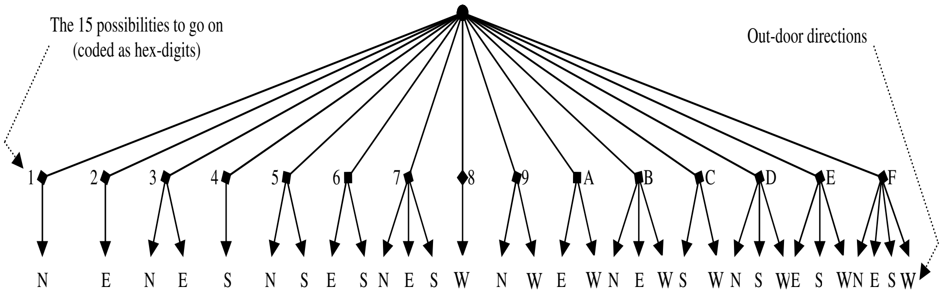

The 15 possibilities to go on
(coded as hex-digits)
Out-door directions
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F
N E N E S N S E S N E S W N W E W N E W S W N S WE S WN E S W

Figure 2. The 2^4-1 variants (coded as hex-digits) to go on from a cell where all possible actions are valid. The digits of the binary representation of 4 bits (from the rightmost one to the leftmost one) of the hex-digits correspond to directions N, E, S and W, respectively

(3^{rd} version) In the followings we consider a third version of the problem that can properly be modeled by an sMDP. The instance that entered the current cell has first to choose among the valid actions to go on. Since the selected action may imply that the current instance has to go on in parallel to more than one direction (by the corresponding doors), it has to make a second decision too: which door will be accessed by it personally (the other doors are randomly distributed among the newly created clones). Given the probability distribution regarding the second choice, determine – for the original agent personally – the most favorable way of getting through our discrete world.

3. Graph representation of Markov decision processes

Stationary MDPs (the dynamics of the model does not depend on time) can be represented as graphs. A dMDP can be conveniently represented as a weighted digraph (see Madani, Thorup and Zwick, 2009). The vertices of the graph correspond to the states of the dMDP and the arcs correspond to the action. The weight of an arc is the immediate cost of the corresponding action.

Since in the case of stochastic sMDPs and g-dMDPs each action may result in several next stages, the digraph representation is not sufficient. As Lew and Mauch (2007) state, an ordinary digraph cannot be used to represent complex systems, such as those with parallel next-states. Two examples described by them are: (1) The main characteristic of the so-called "optimal binary tree problems" is that each decision leads to multiple next-states rather than a single one; (2) Although in probabilistic problems each decision results in a single next-state, these are determined by chance from a set of alternatives.

The common tool for modeling parallel and independent events in an illustrative manner is Petri nets. To model discrete optimization problems that are solvable by DP, Lew (2002) introduced Bellman Nets, special high-level Petri Nets with numerically-colored tokens. Mauch's representation model for similar purposes also relies on specialized Petri Nets that use the standard semantics of place/transition nets, a low-level Petri Net class (see Mauch, 2006). Clempner (2005) introduced the Colored Decision Process Petri Net modeling paradigm that extends the Colored Petri Net theoretic approach including Markov decision processes. As we mentioned above, the Bellman net model cannot handle "cyclic problems" properly, and it has not been extended to MDPs. On the other hand, the Colored Decision Process Petri Net modeling paradigm does not deal with g-dMDPs.

In order to represent a larger class of optimal structure DP problems Katai (2006; 2010) extended the ordinary digraph model firstly to hierarchic d-graphs and later to generalized d-graphs. In this paper we present the way the d-graph representation method can naturally be extended to MDPs (sMDP, dMDP, g-dMDP).

4. d–graph-representation of Markov decision processes

The concept of d-graph was introduced by Katai (2006; 2010).

Definition: The connected weighted bipartite finite digraph Gd (V, E) is a d–graph if:

- $V = V_p \cup V_d$ and $E = E_p \cup E_d$, where
- Vp is the set of p–vertices.
- Vd is the set of d–vertices.
- All in/out neighbours of the p–vertices (excepting the source/sink vertices) are d–vertices. Each d–vertex has exactly one p–in-neighbour. Each d–vertex has at least one p–out-neighbour.
- Ep is the set of p–arcs (from p–vertices to d–vertices).
- Ed is the set of d–arcs (from d–vertices to p–vertices).
- We also define functions C_p: $E_p \rightarrow R$, and C_d: $E_d \rightarrow R$.

If a d–graph is cycle-free, then its vertices can be arranged into levels (hierarchic structure). Related to hierarchic d–graphs Katai (2006) defines the following concepts: d–sub-graph, d–tree, d–sub-tree, d–spanning-tree, optimal d–spanning-tree and optimally weighted d–graph.

A Markov Decision Process (sMDP, dMDP, g-dMDP) can be seen as a d–graph $G = (V_p \cup V_d, E_p \cup E_d)$, where $V_p$is a set of states (p-sources represent initial states and p-sinks goals-states) and $V_d$is a set of actions. Additionally, C_p: E_p →R is a cost (or reward) function, and C_d: E_d →[0,1] is a probability function. A p-vertex has as many d–out-neighbours as the number of actions associated with the corresponding state. A d–vertex has as many p–out-neighbours as the number of next states resulted from the corresponding action. If each d–vertex has exactly one p–out-neighbour and the probabilities associated with d–arcs are 1, then the d–graph represents a dMDP. In the case of g-dMDPs d–vertices may have more than one d–out-arc, each of them associated with probability 1. If a d–graph represents an sMDP, then d–vertices may also have more than one p–out-neighbour. In this case the probability values associated with the d–out-arcs of certain d–vertex represent the probability distribution among the states represented by the corresponding p–out-neighbours. It is assumed that the sum of the probability values associated with the d–out-arcs of a certain d–vertex is 1.

If goal states were defined, the sequence of decisions ends when a goal state is reached. In this case even a finite decision-sequence may be of indefinite (unpredictable) length, terminating when a goal state is reached. We add to the d-graph a dummy goal p-vertex and connect all goal vertices to this one via corresponding d-vertices (all d–in-arcs of the dummy vertex have probability 1). If parallel initial states were defined we add to the d-graph a dummy starting p-vertex and connect it to all source-vertices through one d–vertex. We choose for the p–in-arc of this d–vertex cost/reward 0, and for its d–out-arcs probability 1.

The controller of a dMDP Gselects a finite/infinite length path (containing p– and d–arcs alternatively) that starts at the starting vertex. Infinite length paths contain cycles that are traversed infinitely. The controller of an sMDP or g-dMDPGselects a finite/infinite length d-path. A d-path is a d–sub-graph with a source-vertex and all p-vertices having at most one d–out-neighbour. (Since d-vertices of a d-path may have more than one p–out-neighbours, d-paths may have several branches)

A policy or strategy for a MDP $G = (V_p \cup V_d, E_p \cup E_d)$ is a function π: $V_p \rightarrow V_d$such that for every $v_p \in V_p$(excepting the goal vertex) we have $(v_p, \pi(v_p)) \in E_p$. In other words, a policy corresponds to the selection of one outgoing arc from each p–vertex (such a policy is called stationary, or time invariant). A policy πdefines a finite/infinite path (or d-path) that starts at the starting vertex. Finite paths end at the goal vertex. An infinite path is composed of a (possibly empty) initial path that leads to a cycle which is repeated over and over again. d-paths may have both finite and infinite branches. Policies, as defined above, are Markovian (or memoryless; they depend on history only through the current state) and deterministic.

Given a policy π we define (recursively) the value functions $w_p^\pi : V_p$ →R (also called as cost- or reward-to-go), and $w_d^\pi : V_d \rightarrow$R as follows (for finite horizon problems we have discount factor γ =1, and for infinite horizon problems 0<γ<1)

$$w_p^\pi(v_p) = \begin{cases} C_p\left(v_p, \pi(v_p)\right) + \gamma \cdot w_d^\pi\left(\pi(v_p)\right), & excepting\, the\, goal\, vertex. \\ 0, & for\, the\, goal\, vertex. \end{cases}$$

$$w_d^\pi(v_d) = \sum_{v_p'} C_d\left(v_d, v_p'\right) \cdot w_p^\pi\left(v_p'\right) for\, all\, out - neighbours\, v_p'\, of\, v_d.$$

Since related to infinite branches we have infinite recursion, the discounted (0<γ<1) w_p-values of vertices that are situated along such branches are limits of convergent sequences. The w_p-value of the starting vertex can be considered as an optimality criterion (performance measure) for policy π.

We also define the optimal value functions $w_p^* : V_p \rightarrow$R, and $w_d^* : V_d \rightarrow$R as follows

$$w_p^*(v_p) = opt_{v_d}\left\{C_p(v_p, v_d) + \gamma \cdot w_d^*(v_d) \,|\, for\, all\, out - neighbours\, v_d\, of\, v_p\right\}$$

$$w_d^*(v_d) = \sum_{v} C_d\left(v_d, v_p'\right) \cdot w_p^*\left(v_p'\right) for\, all\, out - neighbours\, v_p'\, of\, v_d.$$

According to the principle of the optimality (the tail of an optimal policy is optimal for the "tail problem") the above optimal value functions implicitly define an optimal policy π^* (for each p–vertex an "optimal d–out-neighbour" is chosen), see Katai (2010). For any given discount factor 0<γ<1, the controller of an infinite horizon MDP always has a single positional policy that produces optimal paths (or d-paths) from every starting vertex (see Madani, Thorup and Zwick, 2009).

5. Combined d–graph algorithm to find optimal policies for finite/ infinite horizon MDPs

In the following we present a combined DP algorithm to find optimal policies for finite/infinite MDPs represented as a d-graph G$(V_p \cup V_d, E_p \cup E_d, C_p, C_d)$.

Computing the w_p^*-value of a p–vertex (excepting the goal vertex) can be implemented as a gradual updating process based on the values of its d–out-neighbours (value iteration method). The starting-value is chosen according to the nature of the optimization problem. The w_d^*-values are recomputed before every use. The w_d^*-values are considered optimal if they are computed from optimal w_p^*-values. Katai (2010) defines the following types of updating operations along p–arcs (if $(v_p, v_d) \in E_p$ and $(w_d^*(v_d)\ + C_p(v_p, v_d))$ is "better" than $w_p^*(v_p)$, then $w_p^*(v_p)$ is updated with value $(w_d^*(v_d)\ + C_p(v_p, v_d))$:

- *Complete*: based on the optimal value of the corresponding d–vertex.
- *Partial*: based on an intermediate value of the corresponding d–vertex.
- *Effective*: effectively improves the value of the corresponding p–vertex.
- *Null*: does not adjust the value of the corresponding p–vertex.
- *Optimal*: sets the optimal weight for the corresponding p–vertex. Optimal updates are complete and effective too.

Katai (2010) presents three d–graph strategies (**d–TOPOLOGICAL**, **d–DIJKSTRA**, **d–BELLMAN–FORD**) that can directly be adopted to the finite MDPs. We extend their application to infinite horizon MDP.

- Function **d–DFS**(G, topo_sequence) inspects if G is cycle free or not, and (if G is cycle free) it establishes the topological order of the vertices (topo_sequence).
- Procedure **d–TOPOLOGICAL**(G, topo_sequence) considers all the vertices according to their reverse topological order and computes their corresponding w_p^*– or w_d^*–values (with γ =1).
- Procedure **d–DIJKSTRA**(G, dijkstra_sequence) applies the Dijkstra strategy in backward way (starting with the goal vertex). This procedure also stores the order the p–arcs were considered (dijkstra_sequence) (with γ =1).
- Function **d–BELLMAN-FORD-tour**(G, dijkstra_sequence, γ) executes updating tours along the dijkstra_sequence of the p–arcs and returns the maximum relative updating value.
- Procedure **d–PRINT-OPTIMAL-POLICY** prints out the optimal path (or d-path) from the starting vertex to the goal vertex.

If the d–graph G is cycle free, then algorithm **d–TOPOLOGICAL** solves the problem most effectively. Otherwise algorithm **d–DIJKSTRA** is applied first. If the first Bellman-Ford-tour (with γ =1) does not detect any updates, this means that **d–DIJKSTRA** has found the optimal solution. Otherwise the Bellman-Ford-tour is repeated until no other update is detected or more than $|V_P|$(the number of p–vertices) tours were performed. If the first repeat-until loop ends because of its second condition this means that the finite horizon MDP does not have an optimal solution and, in this case, the algorithm restarts the Bellman-Ford-tours in their discounted variant (with 0<γ<1). These updating tours are repeated until the wp*-values of all p-vertices are a good enough approximation of their optimal value (with a given ε).

```
ALGORITHM d–optimal-MDP(G, $\gamma$)
  cycle_free ← d–DFS(G, topo_sequence)
  if (cycle_free) then
    d–TOPOLOGICAL(G, topo_sequence)
  else
    d–DIJKSTRA(G, dijkstra_sequence)
    i ← 1
    repeat
      effective_updates ← d–BELLMAN-FORD-tour(G, dijkstra_sequence,1)
        i ← i + 1
    until (effective_updates = 0) or (i >$|V_P|$)
    if (i >$|V_P|$) then //infinite horizon version
```

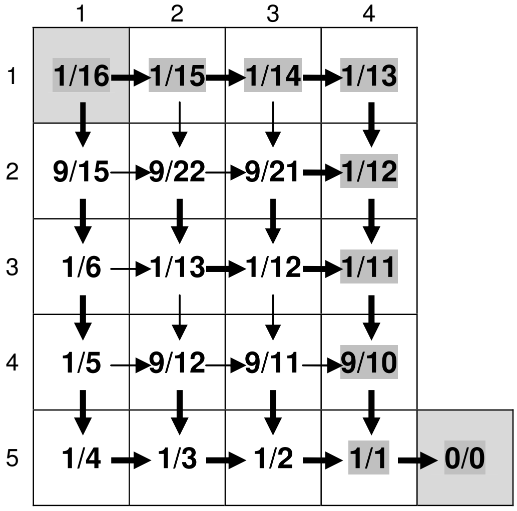
```
        repeat
          effective_updates ← d–BELLMAN-FORD-tour
         (G, dijkstra_sequence, $\gamma$)
        until (effective_updates <$\varepsilon$)
      endif
    endif
    d–PRINT-OPTIMAL-POLICY(G)
endALGORITHM
```

In the following we will illustrate the above strategy on the optimization problem presented in Section 2. The first version of the problem can be modeled by a dMDP. The bi-dimensional world represents the state space (cell (1,1) represents the initial state; we introduced cell (N, M+1) as a dummy goal state). The out-leading doors of each cell represent the possible actions attached to the corresponding state. Each action results with probability 1 in a next state represented by the corresponding neighbor cell. Accordingly, in the attached d–graph all d–vertices have exactly one d–out-arc. We analyze four sub-cases.

- Version 1/a: All doors are oriented from west to east and from north to south (see Fig.3). In this case the attached d–graph is cycle free and the dMDP is solved by procedure **d–TOPOLOGICAL**.
- Version 1/b: Between all neighbor cells there are bidirectional doors (see Fig.4). Since the attached d-graph is cyclic and all immediate costs are positive numbers the dMDP is solved by procedure **d–DIJKSTRA**.
- Version 1/c: The attached d-graph is cyclic, some cells contain negative immediate costs (negative cost means reward), but there are no negative cycles (along which the sum of the immediate cost is negative), see Fig.5. The dMDP is solved by procedure **d–BELLMAN-FORD** (it repeats the procedure **d–BELLMAN-FORD-tour**, with γ =1, until no effective updates are made).
- Version 1/d: The attached d-graph has negative cycles (see Fig.6). The dMDP is solved by the discounted infinite horizon variant of procedure **d–BELLMAN-FORD** (it repeats procedure **d–BELLMAN-FORD-tour**, with γ<1, until no effective updates greater than a given ε are made).

The second version of the problem can be modeled by a g-dMDP (discounted version). Each action may result (with probability 1) in several next states represented by the corresponding neighbor cells (see Fig.7). Accordingly, in the attached d–graph d–vertices may have more than one d–out-arc. In the case of the world presented in Fig. 7 the attached d–graph is cyclic. In this example, although all the immediate costs are positive numbers, some cycles prove to be negative in the sense that (due to the “cloning/splitting and cycling” phenomenon) after each traverse of the cycle the cumulated cost decreases (become lower and lower). Fig. 8 shows the d-path that resulted from applying procedure **d–DIJKSTRA**. Fig. 9 shows the *optimal* d-path that resulted from applying the discounted infinite horizon version of procedure **d–BELLMAN-FORD**. Notice that the optimal path starting with the initial state includes cycles.

The proper model for the third version of the problem is an sMDP. In this case each action results in a single next-state, chosen, though, by chance from a set of alternatives (according to the given probability distribution). As in the previous version of the problem the optimal policy is represented by a d-path, but the controller selects (with a certain probability) an ordinary path for the original agent to follow. Fig. 10 shows the *optimal* d-path that resulted from applying the discounted infinite horizon version of procedure **d–BELLMAN-FORD**. The most probable path of the agent is in bold.



	1	2	3	4	
1	1/16	1/15	1/14	1/13	
2	9/15	9/22	9/21	1/12	
3	1/6	1/13	1/12	1/11	
4	1/5	9/12	9/11	9/10	
5	1/4	1/3	1/2	1/1	0/0

Figure 3. Bi-dimensional world with all doors oriented from west to east and from north to south. The cells contain the corresponding immediate costs and the optimal costs-to-go. Arrows representing the optimal policy are bold. We also marked an optimal path from the initial state to the goal state

6. Conclusion

The major contributions of this paper are: (1) We have extended the definition of dMDP to generalized dMDP (g–dMDP) where each action may result (with probability 1) in more than one next state. Optimization problems characterized by decisions that result in parallel states can properly be modeled by generalized dMDPs; (2) By adapting d-graphs to MDPs we have developed a general tool for representing a larger class of MDPs; (3) We extended the d-graph algorithms described by Katai (2010) to infinite horizon MDPs and we presented a combined DP strategy that, after detecting the characteristics of the attached d–graph, applies the proper DP algorithm that solves optimally the specific problem.

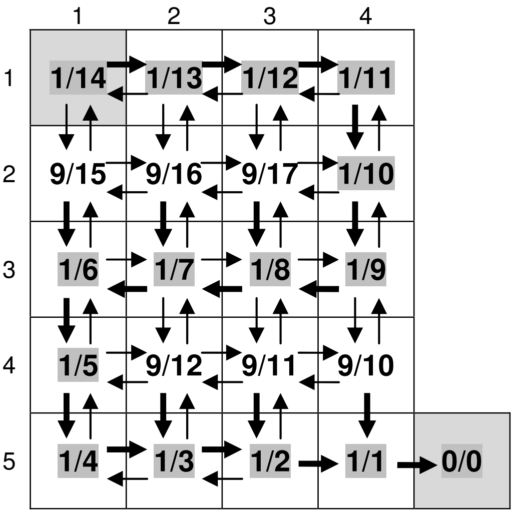


Figure 4. Bi–dimensional world where between all neighbor cells there are bidirectional doors. The cells contain the immediate costs and the optimal costs-to-go. Arrows representing the optimal policy are bold. We also marked the optimal path from the initial state to the goal state

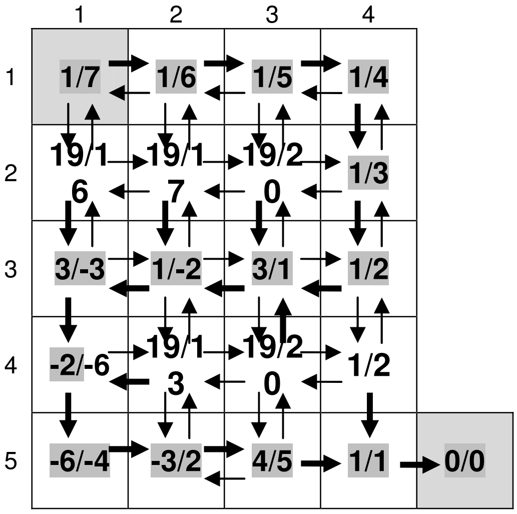


Figure 5. Bi-dimensional world where some cells contain negative immediate costs but there are no negative cycles. The cells contain the corresponding immediate costs and the optimal costs-to-go. Arrows representing the optimal policy are bold. We also marked the optimal path from the initial state to the goal state

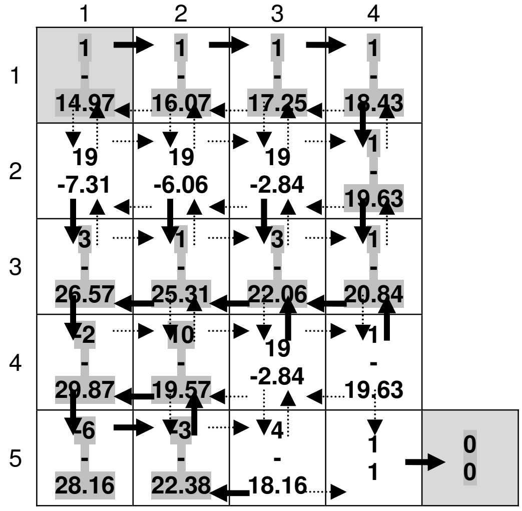


Figure 6. Bi-dimensional world with negative cycles: (4,1), (5,1), (5,2), (4,2), (4,1). The cells contain the corresponding immediate costs and the optimal discounted (γ =0.99) costs-to-go. Arrows representing the optimal policy are bold. We also marked the optimal path starting with the initial state

	1	2	3	4
1	1 (6)	1 (6ACE)	1 (6ACE)	1 (4)
2	9 (3567)	9 (35679ABCDEF)	9 (35679ABCDEF)	1 (459CD)
3	1 (3567)	1 (35679ABCDEF)	1 (35679ABCDEF)	1 (459CD)
4	1 (3567)	9 (35679ABCDEF)	9 (35679ABCDEF)	9 (459CD)
5	1 (2)	1 (239AB)	1 (239AB)	1 (0)

Figure 7. Bi-dimensional world. The cells (representing the states) contain the corresponding immediate costs and the valid actions. The lengths of the hex-strings represent the number of the valid actions and the hex-digits code the directions (and implicitly the next states) resulting from the corresponding action

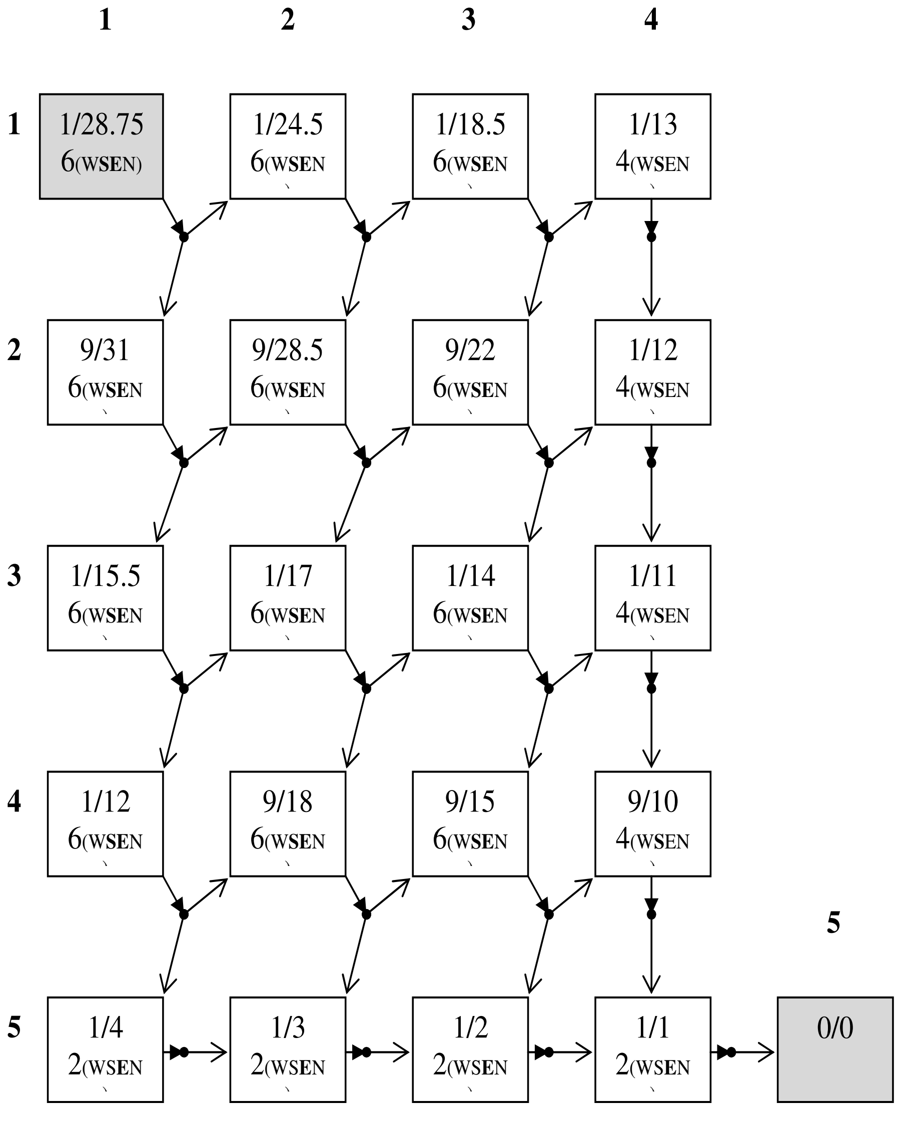


Figure 8. d–path representing the policy obtained from procedure **d–DIJKSTRA.** The first line of each cell contains the corresponding immediate costs and the costs-to-go. The second line of each cell contains the selected hex-digit and the corresponding directions (the bold ones)

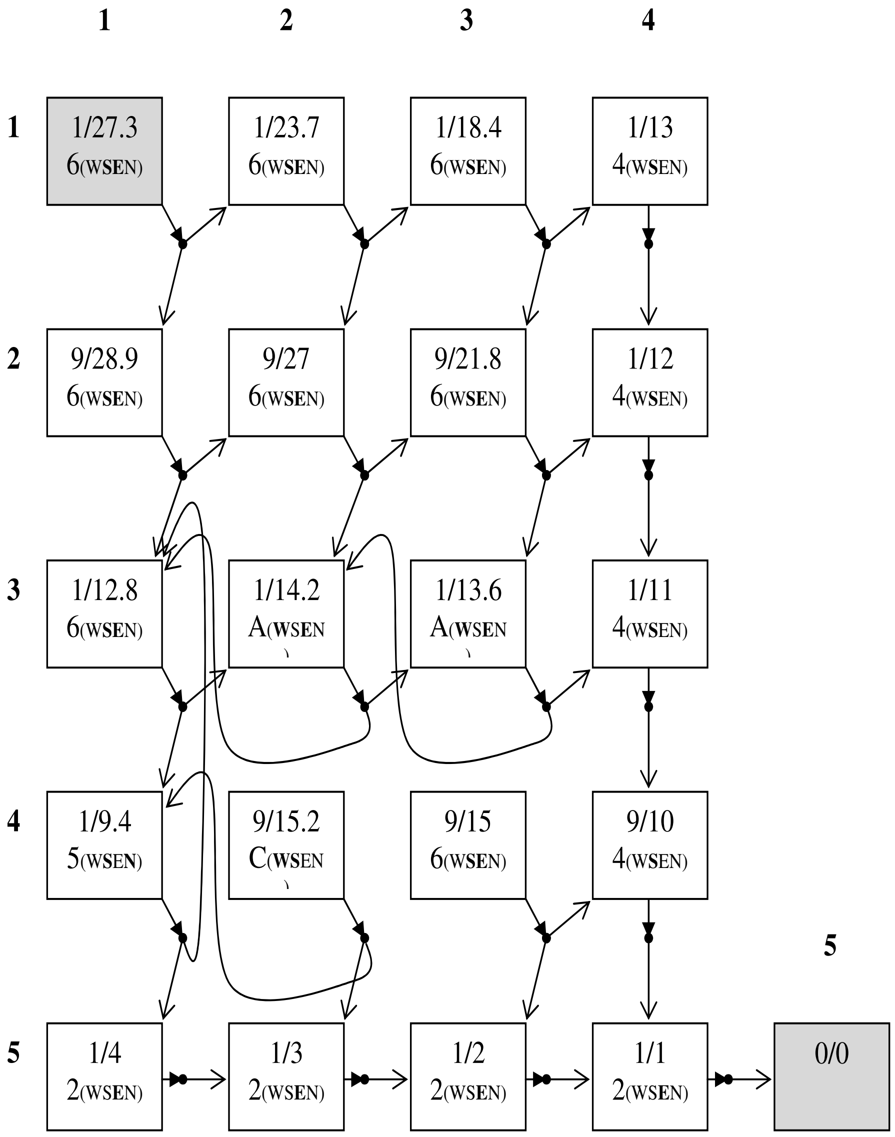


Figure 9. d–path representing the *optimal* policy obtained from the discounted infinite horizon version of procedure **d–BELLMAN-FORD.** The first line of each cell contains the corresponding immediate costs and the *optimal* costs-to-go. The second line of each cell contains the optimal hex-digit and the corresponding directions (the bold ones)

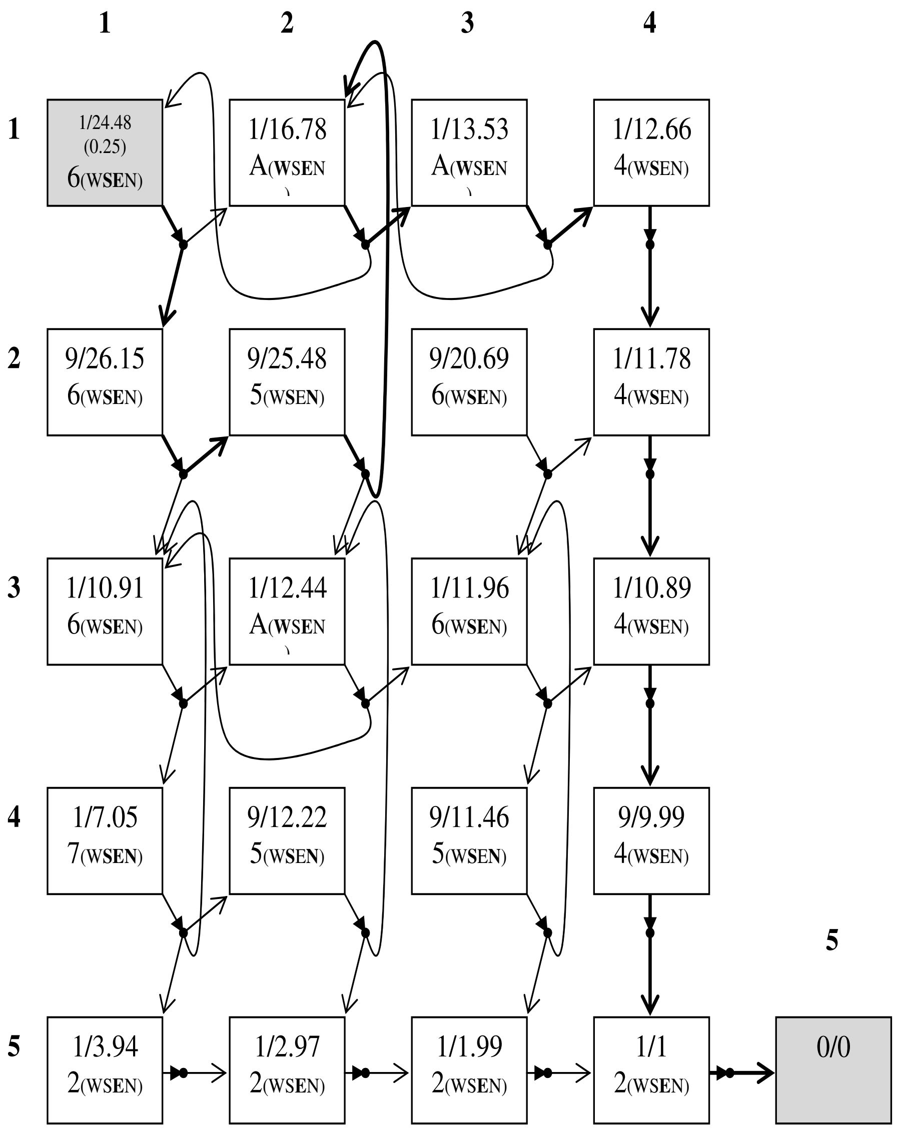


Figure 10. d–path representing the *optimal* policy obtained from the discounted infinite horizon version of procedure **d–BELLMAN-FORD.** The first line of each cell contains the corresponding immediate costs and the *optimal* costs-to-go. The second line of each cell contains the optimal hex-digit and the corresponding directions (the bolded ones). The most probable path for the original agent is bold and has probability 0.25

References

BELLMAN, R. E. (1957) *Dynamic Programming.* Princeton University Press.

CLEMPNER, J. (2005) Colored decision process petri nets: modeling, analysis and stability. *International Journal of Applied Mathematics and Computer Science*, **15** (3), 405–420.

GIMBERT, H. (2008) A Class of Markov Decision Processes with Pure and Stationary Optimal Strategies. *Third Congress of the Game Theory Society.* Evanston, Illinois, USA. Retrieved November 20, 2012, from http://www.labri.fr/perso/gimbert/recherche/games08.pdf

HOWARD, R. A. (1960) *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Massachusetts.

KATAI, Z. (2006) Dynamic programming and d-graphs. *Studia Universitatis Babes-Bolyai, Informatics*, **51** (2), 41-52.

KÁTAI, Z. and CSÍKI, A. (2009) Automated dynamic programming. *Acta Universitatis Sapientiae, Informatica*, **1** (2), 149-164.

KÁTAI, Z. and FÜLÖP, P. I. (2010) Modeling dynamic programming problems: Petri nets versus d-graphs. In: *Proceedings of 8th International Conference on Applied Informatics (ICAI)*, **1**, 217-226. Eger, Hungary.

KATAI, Z. (2010) Modelling dynamic programming problems by generalized d-graphs. *Acta Universitatis Sapientiae Informatics*, **2** (2), 210-230.

KRISTENSEN, A. R. (1996) Dynamic programming and Markov decision processes. *Dina Notat*, **49**. Retrieved November 20, 2012, from Danish Informatics Network in the Agricultural Sciences Web Site: http://www.prodstyr.ihh.kvl.dk/pdf/notat49.pdf

LEW, A. (2002) A Petri net model for discrete dynamic programming. In: *Proceedings of the 9th Bellman Continuum: International Workshop on Uncertain Systems and Soft Computing*, 16–21. Beijing, China.

LEW, A. and MAUCH, H. (2004) Bellman nets: A Petri net model and tool for dynamic programming. In: *Proceedings of 5th International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences (MCO)*, 241-248.

LEW, A. and MAUCH, H. (2007) *Dynamic Programming, A Computational Tool. Studies in Computational Intelligence*, **38**. Springer.

MADANI, O., THORUP, M. and ZWICK, U. (2009) Discounted Deterministic Markov Decision Processes and Discounted All-Pairs Shortest Paths. In: *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, 958-967.

MAUCH, H. (2006) DP2PN2Solver: A flexible dynamic programming solver software tool. *Control and Cybernetics*, **35**(3), 687-702.

PUTERMAN, M. L. (1994) *Markov Decision Processes.* Wiley.