Aggregation-disaggregation algorithms for discrete stochastic systems

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science

Memorandum COSOR 87-30

Aggregation - disaggregation algorithms for discrete stochastic systems

by
Grzegorz Reyman and Jan van der Wal

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Eindhoven, October 1987
The Netherlands

AGGREGATION - DISAGGREGATION ALGORITHMS FOR DISCRETE STOCHASTIC SYSTEMS

Grzegorz Reyman and Jan van der Wal, Eindhoven

Zusammenfassung

In dieser Arbeit wird ein Aggregation - Disaggregation Verfahren vorgestellt für einen Markov Entscheidungsprozess mit entlichen Horizont und zwei-dimensional Zustands- und Aktionsräume. Die zweite Dimension enthält eine gleiche Art von Information und Aggregation hierin ist natürlich und einfach. Die Kwalität des Verfahrens ist illustriert mit einem Beispiel.

Abstract

In this paper an aggregation - disaggregation method is formulated for a finite horizon Markov decision process with two-dimensional state and action spaces. This second dimension of the state and the action contains a similar type of information in which aggregation is both natural and simple. The quality of the approach is illustrated by an example.

1. Introduction

In practice Markov decision models typically give rise to very large state spaces. Then straightforward computation of an optimal strategy is out of the question, but sometimes decomposition or aggregation and disaggregation might work very well. See e.g. Courtois [1], Whitt [5], Mendelssohn [2] and Schweitzer et al. [3].
In this paper we consider a finite horizon Markov decision model with two dimensional state and action spaces, which due to one of the two dimensions are large. Aggregation in this dimension in the action space leeds to the same aggregation in the state space. Typical examples are models with a limited resource which is used for the actions and for which the state contains the remaining amount of resource, or models where in a certain time a total production level has to be achieved. The action contains the production for the next period and the state information about what already has been produced.
A simple example of the latter is the following. In the next 20 hours we have to produce 4288 items on a machine. The production speed of the machine can be varied between 0 and 320 per hour. The machine may however breakdown and the probability of a breakdown increases if the production speed is increased. Repair times can be reduced at additional costs. If at the end of the planning period not all 4288 items are ready, penalty costs are incurred. Assuming the machine breaks down only at the end of an hour, we obtain a fairly standard Markov decision problem.
Let the action set be

$\{ (r,0), (p,0), (p,64), (p,128),..., (p,320) \}$

where r stands for repair and p for produce, then the state space will be

$\{ (d,0), (d,64), (d,128),..., (d,4288), (u,0), (u,64), (u,128),..., (u,4288) \}$

with d denoting that the machine is down and u that it is up. In this case, however, the outcome of the optimization may be poor. On the other hand, if the action set is taken to be

$\{ (r,0), (p,0), (p,1),..., (p,320) \}$,

the state space will be

$\{ (d,0), (d,1),..., (d,4288), (u,0), (u,1),..., (u,4288) \}$,

the computed solution will be optimal but at the cost of an enormous increase in computing time.
It is clear that one has to find a balance between accuracy and computing time, in particular if we realize ourselves that the model used is not a perfect description of reality. In problems of this type an aggregation-disaggregation approach may work well (cf. Veugen et al. [4]).
The remainder of this paper is organized as follows. In Section 2 the model is introduced, Section 3 contains four aggregation-disaggregation algorithms and in Section 4 the performance of the presented algorithms is compared for the 20 hour production planning problem formulated above.

2. Model

We consider a finite horizon Markov decision problem with T periods : $0,1,...,T-1$. The state and action spaces of the problem are two dimensional and denoted by $I\times X$ and $A\times U$ respectively. The sets I and A are small compared to $X=\{0,1,...,N\}$ and $U=\{0,1,...,M\}$. The aggregation will be performed on X and U.
Think of x in the state space pair (i,x) as the used amount of resource or the number of items already produced, and of the u in the action pair (a,u) as the amount of resource used for the execution of a or the number of items to be produced in the next period.
This interpretation of x and u suggests the following simple transition structure. If in the state (i,x) the action (a,u) is taken the system makes a transition to the state $(j,x+u)$ with a probability $p_{ij}^a(u)$, with $\sum_j p_{ij}^a(u)=1$. So only states (j,y) , with $y=x+u$ can be reached. As a result of the action (a,u) in (i,x) there is an immediate reward $r(i,x,a,u)$.
Further a terminal reward $V_0(i,x)$ is obtained if, as a result of the last action, the system is in the state (i,x) at the time N, eg. to cover used resource or shortage.
We denote by $V_T(i,x,\pi)$ the expected reward for the initial state (i,x) when the strategy π is used. Then an optimal Markov strategy can be obtained by the following dynamic programming recursion :

For $n=0,1,...,T-1$ compute for all $(i,x)\in I\times X$

$$V_{n+1}(i,x) = \max_{a,u} \{ r(i,x,a,u)+\sum_j p_{ij}^a(u)V_n(j,x+u) \} \tag{1}$$

The Markov strategy $\pi^* = (f_{T-1}, f_{T-2},..., f_0)$, with $f_n : I\times X \rightarrow A\times U$ and $f_n(i,x)$ maximizing the right hand side in (1), is optimal.
Often at time n only a subset of the states need to be considered. For example in the production problem the initial state is (machine works, 0 produced) thus at time 1 only states (i,x) with $0\leq x\leq 320$ are possible. Also the set of possible actions in a state may be a subset of $A\times U$. For simplicity of

notation we will have the same state and action space for all n .
From the form of the transition probabilities it follows that aggregation in the set U leads to an aggregation in the set X. For example considering only multiplies of k for u means that x will be a multiple of k as well, given it is a multiple of k initially. This is the simplest form of aggregation : restriction to multiples. Choose k and define $\hat{X} = \{0,k,2k,...,\alpha k\}$ and $\hat{U} = \{0,k,2k,...,\beta k\}$ with α (β) the largest integer with $\alpha k \le N$ ($\beta k \le M$) respectively . Solve the recursive scheme (1) with $I \times X$ replaced by $I \times \hat{X}$ and $(a,u) \in A \times \hat{U}$. Sometimes a value for k can be chosen for which the solution is close to optimal and the computing time remains acceptable.
In the next section we present four aggregation-disaggregation algorithms for the case the above simple approach fails.
When describing these algorithms the following notation will be useful,

$$X_l = \{0 , 2^l , 2 \cdot 2^l , 3 \cdot 2^l , \ldots , \alpha \cdot 2^l \}$$

with α the largest integer for which $\alpha \cdot 2^l \le N$. Similarly we define U_l.

3. Aggregation - Disaggregation algorithms

In this section we subsequently consider four aggregation-disaggregation algorithms.

3.1. Algorithm A

The simplest form of aggregation-disaggregation (AD) is AD in the action space only. For each u and for each state (i,x) a separate AD routine is executed to obtain $V_{n+1}^A(i,x)$. For $n = 0, 1, \ldots, T-1$ and $(i,x) \in I \times X$ the value $V_{n+1}^A(i,x)$ is computed as follows.
Consider a sequence of aggregations in the action space, say $U_L, U_{L-1}, \ldots, U_1, U_0$. First compute in the state (i,x) for each a the best $u \in U_L$,i.e., determine

$$\max_{u \in U_L} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^A(j,x+u) \} . \tag{2}$$

Denote this maximizing u by $u_L^A(i,x,a,n)$. Next, compare this u with two neighboring values in U_{L-1} . So, using the notation

$$U_l(u) := \{u-2^l, u, u+2^l\} \cap U_l \quad , \; u \in U_{l+1} , \tag{3}$$

this next step can be written as

$$\max_{u \in U_{L-1}(u_L^A(i,x,a,n))} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^A(j,x+u)\} .$$

Call the maximizer $u_{L-1}^A(i,x,a,n)$ and repeat the procedure for $l = L-2, L-3, \ldots, 1$ by computing

$$\max_{u \in U_l(u_{l+1}^A(i,x,a,n))} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^A(j,x+u)\} .$$

And finally we obtain

$$V_{n+1}^A(i,x) = \max_a \max_{u \in U_0(u_1^A(i,x,a,n))} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^A(j,x+u)\} . \tag{4}$$

Denote this maximizing pair by $(a,u)^A(i,x,n)$. Once we have computed the $V_T^A(i,x)$ for all, or merely

the initial, (i,x) we have a supposedly optimal value and the various $(a,u)^A(i,x,n)$ together constitute the supposedly optimal strategy.
Two remarks have to be made. First note that the u_l^A need not to be unique. So we need a tie-break rule, for instance pick the smallest one. Secondly, and far more important, we see that this algorithm will not necessarily find the optimal value and optimal strategy, since the procedure described above may only find a local optimum in some sense. It is not guaranteed that the obtained strategy is nearly optimal. However, in most realistic examples the behavior of the various cost and probability functions is more or less continuous. So it is likely that if we take a modest value of L to begin with, the algorithm will work well. An attempt to safeguard against bad solutions is to extend the sets $U_l(u_{l+1}^A(i,x,a,n))$ with other values of u around values which were not too bad in the previous step. We will come back to this idea in algorithm D.

3.2. Algorithm B

This second AD algorithm can be seen as a repeated version of algorithm A for various aggregation levels in the state space. In the first step the problem is considered to have state space $I \times X_L$ and action space $A \times U_L$. In the second step the problem has state space $I \times X_{L-1}$ and action space $A \times U_{L-1}$ and is solved by algorithm A starting with action space $A \times U_L$. In the final, L-th, step algorithm A is executed with state space $I \times X$ and $A \times U$ as action space . Hopefully not all steps have to be executed. We intend to stop as soon as the computed optimal values $V_T^{B,l-1}$ and $V_T^{B,l}$ for the given initial state (i_0,x_0) are sufficiently close. (It is assumed that $x_0 \in X_l$ for all l).
Formally B runs as follows :
Step 1

Calculate for $n=0,1,\ldots,T-1$ and all $(i,x) \in I \times X_L$

$$V_{n+1}^{B,1}(i,x) = \max_a \max_{u \in U_L} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^{B,1}(j,x+u) \} \tag{5}$$

until we get $V_T^{B,1}(i_0,x_0)$.

Step l, $l=2,\ldots$ (algorithm A with state space $I \times X_{L-l+1}$ and action space $A \times U_{L-l+1}$)

For $n=0,1,\ldots,T-1$ and $(i,x) \in I \times X_{L-l+1}$ compute $V_{n+1}^{B,l}(i,x)$ as follows.
For each a first determine

$$\max_{u \in U_L} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^{B,l}(j,x+u) \} \tag{6}$$

and denote the minimizer by $U_L^{B,l}(i,x,a,n)$.
Next compute for $m=L-1,\ldots,L-l+2$

$$\max_{u \in U_m(u_{m+1}^{B,l}(i,x,a,n))} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^{B,l}(j,x+u) \} . \tag{7}$$

And finally

$$V_{n+1}^{B,l}(i,x) = \max_{a} \max_{u \in U_{L-l+1}(u_{L-l+2}^{B,l}(i,x,a,n))} \{ \, r(i,x,a,u) + \textstyle\sum_j p_{ij}^{a}(u) V_n^{B,l}(j,x+u) \, \} \; . \tag{8}$$

Stop
if $V_T^{B,l-1}(i_0,x_0)$ and $V_T^{B,l}(i_0,x_0)$ are sufficiently close, where (i_0,x_0) is the initial state.

3.3. Algorithm C

C can be seen as a refinement of B. Instead of merely a repetition of A's, as B is, we also use the information about which values of u were good for a, that has been obtained in the previous step. Formally C runs as follows.
Step 1

Calculate for n=0,1,...,$T-1$, for all $(i,x) \in I \times X_L$ for each a

$$\max_{u \in U_L} \{ \, r(i,x,a,u) + \textstyle\sum_j p_{ij}^{a}(u) V_n^{C,1}(j,x+u) \, \} \tag{9}$$

and denote the maximizer by $u_L^C(i,x,a,n)$.

Compute

$$V_{n+1}^{C,1}(i,x) = \max_{a} \max_{u \in U_L} \{ \, r(i,x,a,u) + \textstyle\sum_j p_{ij}^{a}(u) V_n^{C,1}(j,x+u) \, \} \tag{10}$$

Step l , l=2,3,...

First construct for each n=0,1,...,$T-1$ and each triple $(i,x,a) \in I \times X_{L-l+1} \times A$ the set $U_{L-l+1}^{C}(i,x,a,n)$ of possible values for u in the maximization. If $x \in X_{L-l+2}$ then

$$U_{L-l+1}^{C}(i,x,a,n) = U_{L-l+2}(u_{L-l+2}^{C}(i,x,a,n)) \tag{11}$$

If, however, $x \in X_{L-l+2}$ then $u_{L-l+2}(i,x,a,n)$ has not been determined in the previous iteration. Therefore the information is used that has been obtained for the neighbors of (i,x) : $(i,x-2^{L-l+1})$ and $(i,x+2^{L-L+1})$. So for $x \in X_{L-l+2}$

$$\begin{aligned} &U_{L-l+1}^{C}(i,x,a,n) \\ &\quad := U_{L-l+1}(u_{L-l+2_C}(i,x-2_{L-l+1},a,n)) \bigcup U_{L-l+1}(u_{L-l+2}(i,x+2_{L-l+1},a,n)) \end{aligned} \tag{12}$$

Next compute for each a

$$\max_{u \in U_{L-l+1}^{C}(i,x,a,n))} \{ \, r(i,x,a,u) + \textstyle\sum_j p_{ij}^{a}(u) V_n^{C,l}(j,x+u) \, \} \tag{13}$$

and denote the maximizing u by $u_{L-l+1}(i,x,a,n)$.

Finally compute

$$V_{n+1}^{C,l}(i,x) = \max_{a} \max_{u \in U_{L-l+1}^{C}(i,x,a,n)} \{ r(i,x,a,u) + \sum_j p_{ij}^{a}(u) V_n^{C,l}(j,x+u) \} . \tag{14}$$

Stop if $V_T^{C,l-1}(i_0,x_0)$ and $V_T^{C,l}(i_0,x_0)$ are sufficiently close.

None of these three algorithms is guaranteed to produce an optimal or even nearly optimal solution. As argued before the structure of the problem is such that one might hope for a reasonably good solution provided the level of aggregation L is not too large to begin with. The Figure 1 shows the major problem for the method.

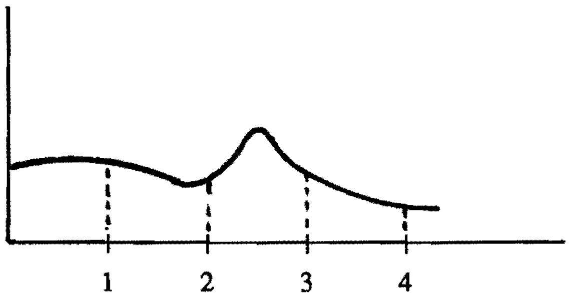


Figure 1

If at a certain stage the points indicated as 0,1,...,4 are calculated, the optimal one seems to be 1. From then on only the interval (0,2) will be considered whereas the optimal value lies between 2 and 3 !
To diminish the risk of making mistakes of this kind we suggest to increase the sets wherein we search for the best values of u. In algorithm D below this is done by taking into consideration also neighborhoods of u's which were only nearly optimal in the previous step. We formulate D as an extension of algorithm C.

3.4. Algorithm D

Algorithm D is executed in exactly the same way as algorithm C with the exception that the sets U^C defined in (11) and (12) are constructed differently.

Step 1

Calculate for $n=0,1,\ldots,T-1$, for all $(i,x) \in I \times X_L$ for each a

$$\max_{u \in U_L} \{ r(i,x,a,u) + \sum_j p_{ij}^{a}(u) V_n^{D,1}(j,x+u) \} \tag{15}$$

and construct the set $U_L^{D,2}(i,x,a,n)$ of values for u which nearly maximize (15). Nearly means that they come within ε_1 of the optimum.

Compute $V_{n+1}^{D,1}$ as in (10).

Step l , l=2,3,...

First construct the sets $U_{L-l+1}^{D,1}(i,x,a,n)$ of values of u that have to be considered.

For $x \in X_{L-l+2}$ define

$$U_{L-l+1}^{D,1}(i,x,a,n) = \bigcup_{u \in U_{L-l+1}^{D,2}(i,x,a,n)} U_{L-l+1}(u) \tag{16}$$

For $x \in X_{L-l+2}$ define

$$U_{L-l+1}^{D,1}(i,x,a,n) = U_{L-l+1}^{D,1}(i,x-2^{L-l+1},a,n) \cup U_{L-l+1}^{D,1}(i,x+2^{L-l+1},a,n). \tag{17}$$

Next compute for each a

$$\max_{u \in U_{L-l+1}^{D,1}(i,x,a,n))} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^{D,l}(j,x+u) \} . \tag{18}$$

and construct the sets $U_{L-l+1}^{D,2}(i,x,a,n)$ of actions which maximize (18) within ε_l . $U^{D,2}$ will be a subset of $U^{D,1}$.

Finally compute $V_{n+1}^{D,l}$ analogously to (14)

$$V_{n+1}^{D,l}(i,x) = \max_a \max_{u \in U_{L-l+1}^{D,1}(i,x,a,n)} \{ r(i,x,a,u) + \textstyle\sum_j p_{ij}^a(u) V_n^{D,l}(j,x+u) \} . \tag{19}$$

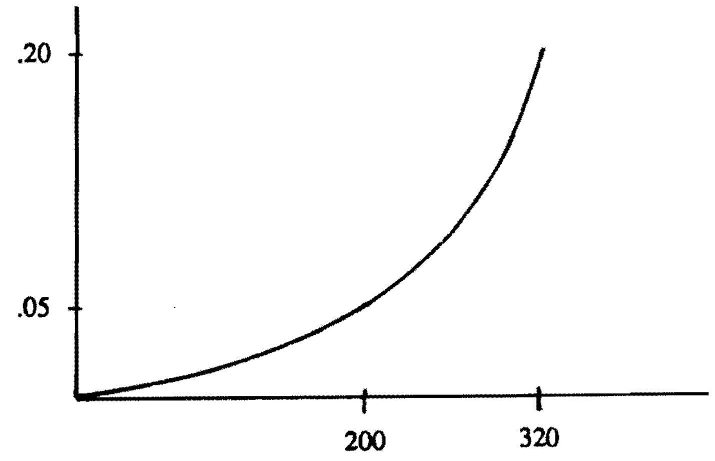
4. Numerical results

The four algorithms have been tested on the production problem mentioned in the introduction : producing 4288 items in 20 hours. The detailed description of the problem is as follows.

- The state set $I = \{0,1\}$ where 0 indicates that the machine is down and 1 that it is available for production
- If the machine is down there are 3 actions, 0 : do nothing, 1 : normal repair, 2 : fast repair
- In state 1 there is only one action, 0 : produce, but the production rate can be varied between 0 and 320 per hour.

- If the machine is down, the costs of the normal repair are 15, the costs of the fast repair 60. The probability that at the end of the hour the repair turns out to be successful is 1/5 for a normal repair and 1/2 for a fast repair.

- The failure probability $p_{10}^{0}(u)$ depending on the production rate is given in Figure 2



$$p_{10}^{0}(u) = \frac{u}{8000-20u}$$

- The shortage costs at the end of the planning period are 2 per item. If at the end of the 20 hours the machine is down there is a terminal cost of 20.

When presenting the results, two things are important : the accuracy of the final result and the computing time needed to get this result.
To obtain the exact solution we solved the problem with X and $U=\{0,1,2,...,4288\}$. In the aggregation we used grid sizes 64,32,16,8,4,2 and 1 if necessary.
The number of states in X_l varies as follows

l	0	1	2	3	4	5	6
Number of states in X_l	4289	2145	1073	537	269	135	68

The exact result :

Minimal cost 480.08

Computing time 3142.4
(All computing times are given in seconds on a VAX 11/750).
The results for the algorithms A,B,C and D are as follows.

grid	Minimal costs A	B	C	D
64	500.03	500.03	500.03	500.03
32	486.54	486.54	486.54	486.54
16	481.22	481.22	481.22	481.22
8	480.37	480.37	480.37	480.37
4	480.17			
2	480.09			
1	480.08			
total computing time	552.5	63.5	18.9	31.2

In algorithms B,C and D we stopped when the difference between two successive iterations was less than 1%. In D the sets of nearly optimal u values consisted of these actions for which the relative difference from optimality was within 1%.
The complete, exact, solution of the problem with state space $I \times X_L$ and action space $A \times U_L$ for $L = 6,5,4$ and 3 (grids 64,32,16 and 8) gave exactly the same result as A,B,C and D in a computation time 80.6.
As we see from the results, the structure of the problem is such that the fact that the algorithms check only a subset of all u values in the grid does not lead to any loss of optimality. Clearly, one may come up with cases which the problem, signalled in Figure 1, of looking in the wrong area, does give an increase in costs.

5. Conclusions

We have presented four aggregation-disaggregation algorithms for Markov decision processes in which aggregation in the action space leads to the same level of aggregation in the state space.
The algorithms have been tested on a simple production problem. Although more numerical experience is wanted we are convinced that in particular algorithm D is a very accurate and fast heuristic.

6. References

[1] Courtois, P.-J. (1977), Decomposability: Queueing and Computer System Application, Academic Press, New York.

[2] Mendelssohn, R. (1982), An iterative aggregation procedure for Markov decision processes, Operations Reseach 30, 62-73.

[3] Schweitzer, P.J., Puterman, M. and K.W. Kindle (1981), Iterative aggregation - disaggregation procedures for solving discounted semi-Markovian reward processes, Working paper series, No. 8123, Graduate School of Management, University of Rochester.

[4] Veugen, L.M.M., Van der Wal, J. and J. Wessels (1985), Aggregation and disaggregation in Markov dicision models for inventory control, EJOR 20, 248-254.

[5] Whitt, W. (1978,1979), Approximations of dynamic programs I and II, MOR 3, 231-243 and 4, 179-185.