COMPUTER DEMONSTRATION: A GENERAL SOFTWARE SYSTEM FOR DYNAMIC PROGRAMMING.

A. R. KRISTENSEN

The Royal Veterinary & Agricultural University, Copenhagen, Denmark.
E-mail: ark@dina.kvl.dk

ABSTRACT

A general software system for construction, editing and optimization of dynamic programming models ranging from traditional models and hierarchic Markov processes to multi-level hierarchic processes is presented. The software has a dual purpose. Firstly it is supposed to fulfill the need of the apprentice for demonstration of methods and concepts and secondly it is to some extend expected to cover the needs of a professional user building real world models. In both cases the visual model interface of the system has turned out to be important.

INTRODUCTION

Agricultural management science has a long and strong tradition for application of dynamic programming and in particular Markov decision processes for decision support. For a survey of studies, reference is made to Kennedy (1986). One of the most frequently studied problems in literature has been the animal replacement problem (Kristensen, 1994). From a theoretical point of view, the development of hierarchic Markov processes (Kristensen, 1988) and multi-level hierarchic Markov processes (Kristensen & Jørgensen, 2000) has widened the applicational scope of the techniques. Even though the methods are obvious for solving sequential decision problems under risk and numerous applications have been described in literature, the lack of standard software is considered to be a true bottleneck for a more widespread use.

The lack of standard software is a limitation in two senses. Firstly, a software package with a good user interface is important for the comprehension of the method among students and young scientists who may easily construct small models serving as examples. Secondly, when it comes to real world models, it requires a rather high level of experience with software construction to build a working prototype if no standard software is available. This applies in particular, when the more sophisticated algorithms like policy iteration (Howard, 1960) and hierarchical algorithms are used. That is probably the reason that most studies reported in literature have used the more straight forward value iteration algorithm (Howard, 1960).

The benefits of a good software system become obvious if we turn our attention to the area of Bayesian Networks (Jensen, 1996) where the Hugin™ software system through a very illustrative graphical user interface demonstrates the basic concepts of the method in an excellent way to the apprentice and provides a comprehensive application programming interface for the professional user. The only attempt known to the author to make a standard software system for Markov decision processes was done by Kennedy (1986) who supplied the so-called GPDP (General Purpose Dynamic Programming) software written in Basic.

Unfortunately, the GPDP software neither fulfills the need for demonstration purposes nor the needs of a professional user.

A software system called MLHMP (Multi-Level Hierarchic Markov Processes) has been developed. MLHMP is supposed to fulfill the need of the apprentice for demonstration, and also to some extent the needs of a professional user (very large models still need software specifically designed for the problem in question).

THE MLHMP SOFTWARE

Facilities of the programme

The MLHMP software has a graphical user interface for construction and editing of multi-level hierarchic Markov processes. Since ordinary Markov decision processes as well as bi-level hierarchic processes are just special cases, such models may be handled as well. As illustrated in Figure 1, models are shown in a tree-like hierarchic structure, where the founder process always is displayed as the root node at the upper left corner of the window. A node of the tree is either a process, a stage, a state or an action. The child nodes of a process are stages and child nodes of a stage are states etc. An action is either a leaf node which means that the parameters (reward, output, transition probabilities) are defined directly, or it has one and only one child which accordingly is a child process. A node may display as expanded (showing its children) or collapsed. A mouse click changes the state of the node from collapsed to expanded and vice versa.

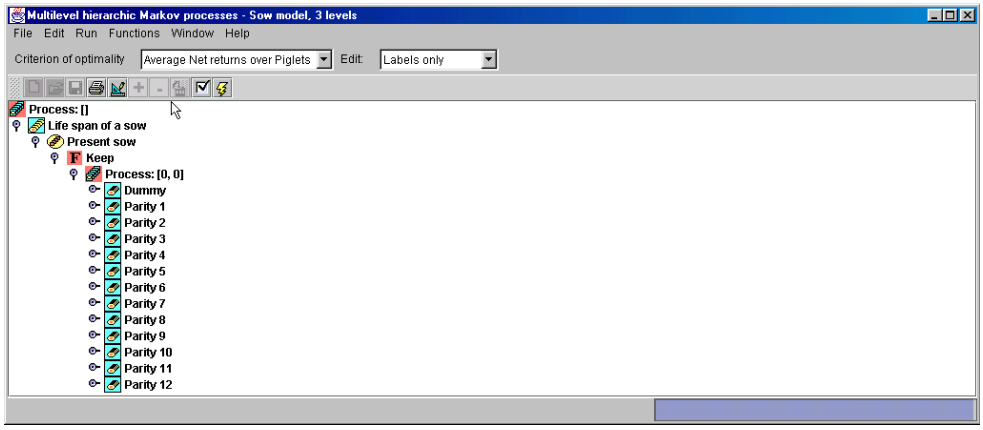
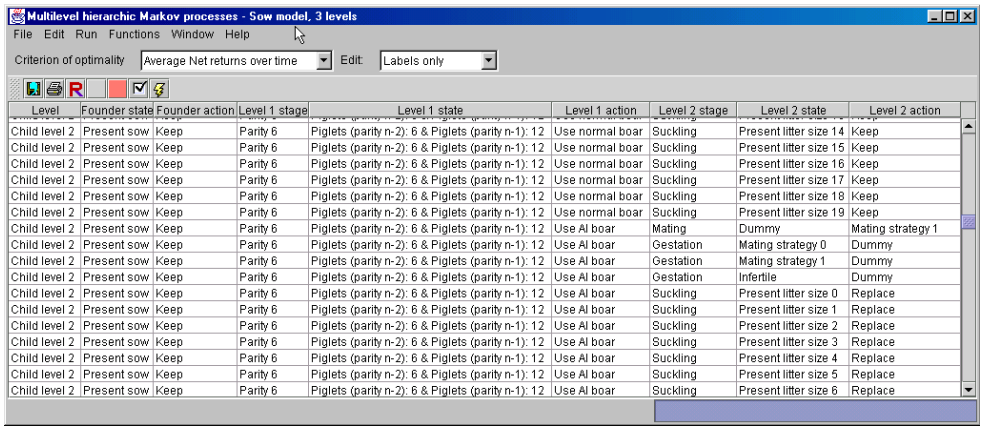


FIGURE 1. The “Process tree” window of the MLHMP software.

It is possible to build models from scratch and to edit existing models using this interface to successively add processes, stages, states and actions until the structure of the model is finished. In order to transform the graphic tree image to a working model, a compilation creating the model is necessary. After compilation, the parameters of the model may be entered using a specific node editor window. Having entered all parameters, the model may be optimized by policy iteration or value iteration as desired. Optimal policies may be calculated under three different criteria of optimality. The value of the objective function under the optimal policy as well as a log of the iteration may be inspected in a separate ”Optimization log” window (not shown). The optimal policy is displayed as a large table in a separate “Result” window (Figure 2). The table may also show the present value (or - depending on the criteria of optimality - the relative value) of all actions.

A probabilistic simulation facility is available in order to investigate the consequences of non-optimal policies. Alternative policies may be specified in the "Result" window, and the consequences calculated by Markov chain simulation are shown in the "Optimization log" window. Furthermore, additional technical and economical key figures may be calculated under optimal (or arbitrarily defined) policies. Finally models may be saved and loaded to/from disk.



Level	Founder state	Founder action	Level 1 stage	Level 1 state	Level 1 action	Level 2 stage	Level 2 state	Level 2 action
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use normal boar	Suckling	Present litter size 14	Keep
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use normal boar	Suckling	Present litter size 15	Keep
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use normal boar	Suckling	Present litter size 16	Keep
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use normal boar	Suckling	Present litter size 17	Keep
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use normal boar	Suckling	Present litter size 18	Keep
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use normal boar	Suckling	Present litter size 19	Keep
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Mating	Dummy	Mating strategy 1
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Gestation	Mating strategy 0	Dummy
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Gestation	Mating strategy 1	Dummy
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Gestation	Infertile	Dummy
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Suckling	Present litter size 0	Replace
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Suckling	Present litter size 1	Replace
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Suckling	Present litter size 2	Replace
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Suckling	Present litter size 3	Replace
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Suckling	Present litter size 4	Replace
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Suckling	Present litter size 5	Replace
Child level 2	Present sow	Keep	Parity 6	Piglets (parity n-2): 6 & Piglets (parity n-1): 12	Use AI boar	Suckling	Present litter size 6	Replace

FIGURE 2. The "Results" window of the MLHMP software.

TECHNICAL DESCRIPTION

The MLHMP software is programmed entirely in Java. It is therefore independent of platform. In practice it has been tested under Microsoft Windows 95/98, IBM OS/2 Warp 4.0 and Unix (Sun Solaris). It is free and may be downloaded and installed directly from the internet1. A Users' Guide is available (Kristensen, 1999).

All model elements are represented as object classes. An entire model is represented as an array of levels, where each level in turn is an array of processes. Basically a process is represented as an array of stages, a stage as an array of states, and a state as an array of actions supplemented by fields for selected action and the value function under value iteration (i.e., fi(n) of Eq. (1) and similar expressions for expected reward and output until the end of the current process):

$$f_i(n) = \max_d \left\{ r_i^d(n) + \sum_j p_{ij}^d \beta_{ij}^d f_j(n+1) \right\}, \forall i \le I, \forall n < N \qquad (1)$$

where rid(n) is the reward under action d of state i at stage n, pi1d(n),…, piJd(n) are the corresponding transition probabilities, and βi1d(n),…, βiJd(n) are the discount factors.

An action in turn is an object class defined as

```
public class StateInstant

      static int rewardFrom = 0;

      static int outputFrom = 1;
```

[1] http://www.prodstyr.ihh.kvl.dk/software/mlhmp.html

```
Object discount;

float quantities[];

ProbabilityProvider transition;

float result;

Stage next_stage;

Process child;

boolean terminator;

String label;

MutableTreeNode parent;
```

where the fields rewardFrom and outputFrom are static (i.e. global) variables defining the position of the reward and output elements, respectively, of the quantities array which at other positions may contain other quantities describing the properties of the model. The idea is to make it possible to redefine the interpretation of reward and output by assigning new values to rewardFrom and outputFrom. The field discount contains the stage length(s). It may either be an array of float values or just a single Float value depending on whether or not stage length depends on next state. The result field contains the value in brackets on the right hand side of Eq. (1), i.e. before maximization over actions. The next_stage field is a pointer to next stage; the child field is either null (at the bottom level) or it contains a pointer to the relevant child process. The terminator field has the value true if the action may terminate the current process before the final stage (this facility will be illustrated later). The label field is just an explanatory text string, and the parent field is necessary for drawing the process tree image shown in Figure 1.

The most sophisticated field of an action (or a StateInstant as it is denoted in the programme code) is the transition field which has to be an instance of a subclass of the abstract ProbabilityProvider class. Any extended class must define a method get(j) returning the probability of observing state j at the next stage (which in turn is known from the next_stage field). This very general representation of the transition probabilities has many advantages in particular in relation to memory needs of the model. The default ProbabilityProvider class is a so-called ArrayProbabilityProvider class just representing the transition probabilities as a simple array. Other possible ProbabilityProvider subclasses generate deterministic transitions (where one state j' has the probability 1 and all other have the probability zero), sparse transitions (where only a few directly defined future states have non-zero probabilities), normal distributions, poisson distributions or binomial distributions. Other subclasses may be defined as convenient. The choice of the most appropriate ProbabilityProvider is often a tradeoff between memory and time. If the standard ArrayProbabilityProvider is used, the relevant probability is always available, but if next_stage has many states, this way of representation is very memory consuming. If J denotes the number of states, the ArrayProbabilityProvider stores J float values, whereas the deterministic subclass only stores 1, and the other distributions typically two. On the other hand, the latter methods require that a transition probability is calculated each time it is needed. The user interface lets the user select the desired distribution from a pull-down menu and prompts for entering of necessary parameters in an popup window. If the “true” distribution is one of the predefined standard

distributions, it is probably most convenient to specify them directly in terms of their parameters (e.g. mean and standard deviation for a normal distribution). Optionally, they may afterwards be converted to simple arrays in order to speed up the calculations.

The basic operations like simulation and optimization by value iteration or policy iteration are defined as methods at model level, but, involving many methods defined at other levels. For instance, at process level, a method called determinePolicy determines a new policy for a process given a specific terminal or average value. At action level, the method getParameters collects the reward, output and transition probabilities from a child process.

Even though models may be built manually from scratch using the graphical user interface, such a procedure is an extremely tedious task for real world problems. Instead, such models should be defined as a separate model class extending the abstract ModelProvider class. Such classes may be considered as a kind of plugins to the MLHMP software, and they must implement the method getHierarchicModel which returns an entire model. The "New" menu item of the "File" menu will display a list of properly installed plugins. Selecting a specific plugin will generate the corresponding model. It is entirely up to the plugin to decide in what way the model is created. It may be as a "batch job" with everything decided in advance, or it may be through an interactive dialog with the user (for instance prompting for prices, correlations, etc.). An example of a plugin building a sow replacement model is described by Kristensen (2000).

DISCUSSION

A Markov decision process is a rather abstract concept. In particular the multi-level variant may be difficult to grasp and really learn to understand. One of the most important benefits of the program is to make models directly visible through the graphical user interface available as the process tree window (Figure 1). This user interface may be used directly by the apprentice to easily construct and run small examples from scratch. In that way it is easy to obtain a general understanding of the structure and parameter needs of such models - a knowledge which is essential for application of the method for real world problems. This "demonstration" effect was the original purpose of the process tree window.

It has, however, later become clear that the process tree window is also a very valuable tool for the professional user. Even though it is prohibitive to build real world models from scratch

in the process tree window, it serves as a model browser transforming a very abstract concept to a directly visible model. Even though construction of a real world model requires that a new plugin is programmed, the programmer may use the process tree window to check that the structure of the model becomes exactly as desired. The same applies to parameter values. The plugin defines the parameters, but afterwards the values may be checked in the process tree window. Even with a rather large model, it only takes a few mouse clicks to take the user to exactly the part of the model where he wants to check structure or parameters. In other words, the process tree window serves as an important tool for internal model validation.

The results window (Figure 2) is ideal for small demonstration models, but the table is very large for real world models. Nevertheless, the program is able to handle even big tables, and if the present or relative values are displayed instead, the number of rows is almost doubled. For any practical analysis of results, the contents of the table have to be exported to a text file and

reloaded in a statistical tool. It should be noticed, that exactly the same information is available by clicking the appropriate nodes of the process tree.

The MLHMP software is a general tool using an object oriented design. It is therefore very flexible and may represent any ordinary or hierarchic Markov process as long as time, state spaces and action spaces are discrete. The generality is to some extend in conflict with efficiency. If a software program is constructed directly for a specific model it will almost always be possible to take a special structure into account and thereby improve efficiency. Many features of MLHMP are nevertheless available in order to increase efficiency (for instance deterministic transitions, directly defined transition probabilities of parent processes, and an ability to share child processes if they only differ concerning the initial distribution.

The example described by Kristensen (2000) is of intermediate size. It corresponds very much to dairy cow replacement models published in the eighties. It is not assumed that the MLHMP software is able to handle very big models like Houben et al. (1994).

REFERENCES

Houben, E. H. P., R. B. M. Huirne, A. A. Dijkhuizen, and A. R. Kristensen (1994). Optimal replacement of mastitis cows determined by a hierarchic Markov process. *Journal of Dairy Science*, 77:2975-2994.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. The M.I.T. Press, Cambridge, Massachusetts.

Jensen, F. V. (1996). *An Introduction to Bayesian Networks*. UCL Press, London, 1996.

Kennedy, J. O. S. (1986). *Dynamic Programming Applications to Agriculture and Natural Resources.* Elsevier Applied Science Publishers, London and New York.

Kristensen, A. R. (1988). Hierarchic markov processes and their applications in replacement models. *European Journal of Operational Research*, 35:207-215.

Kristensen, A. R. (1994). A survey of Markov decision programming techniques applied to the animal replacement problem. *European Review of Agricultural Economics*, 21:73-93.

Kristensen. A. R. (1999). Software users' guide: Multi-level hierarchic Markov processes. *Dina Notat* 84, Danish Informatics Network in the Agricultural Sciences, Copenhagen, September 1999.

Kristensen, A.R. (2000). A software system for multi-level hierarchic Markov processes. *Proceedings of the International Symposium on Pig Herd Management Modeling and Informations Technologies Related.* September, 18-20. Lleida, Spain.

Kristensen, A. R. and Jørgensen, E. (2000). Multi-level hierarchic Markov processes as a framework for herd management support. *Annals of Operations Research*, 94:69-89. 2000.