Modeling with LIMIDs - Examplified by Disease Treatment in Slaughter Pigs

Michael Höhle*
hoehle@dina.kvl.dk

Erik Jørgensen†
Erik.Jorgensen@agrsci.dk

Dennis Nilsson*‡
nilsson@math.auc.dk

Abstract

This paper shows the applicability of LImited Memory Influence Diagrams (LIMIDs) for modeling pig production decision scenarios. Contrary to influence diagrams LIMIDs do not require remembering of previous information in multi-stage decision problems. An easy LIMID evaluation algorithm using existing software packages is presented. Furthermore, we adopt the techniques from Object Oriented Bayesian Networks (OOBNs) to LIMIDs resulting in the concept of Object Oriented LIMIDs (OOLs). With the OOL-tool we are one step closer to a coherent process of object oriented analysis, design and implementation of multi-stage decision scenarios. OOL modeling is illustrated through an example from pig production, where a veterinarian on a weekly basis has to decide about treatment of pigs in slaughter pens.

1 Introduction

Handling multi-stage decision scenarios in partially observable stochastic domains is an essential technique for mathematic-economic modeling of pig production. Using influence diagrams (IDs) [4] to handle especially dynamic decision scenarios from pig production suffers from notorious complexity problems due to the assumption of no-forgetting, that is information available for a decision must also be available for any later decision. Besides resulting in complexity problems, this assumption often results is incorrect models of the decision scenario. An example is treatment of slaughter pigs by a veterinarian on weekly visits. No records of the individual animal exists, thus information used for earlier decisions are unavailable to a new or forgetful veterinarian the following week. A different approach is the use of Partial Observed Markov Decision Processes (POMDPs), see [6]. Here, the underlying process is assumed Markov, however the true state of the system is only partially observed. The Markov assumption is often reasonable, but what we would really like is a flexible modeling language which allows us to model exactly what information persists at which point in the multi-stage decision problem. Inspired by the framework of object oriented programming (OOP) the modeling language should be component based allowing interleaved top-down and bottom-up modeling by enabling encapsulation, polymorphic capabilities etc. Such a modeling language is a necessity for

* Department of Animal Science and Health, Royal Veterinary and Agricultural University of Denmark, Denmark

† Danish Institute of Agricultural Sciences, Department of Agricultural Systems, Biometric Research Unit, Research Centre Foulum, Denmark

‡ Department of Mathematical Sciences, Aalborg University, Denmark

systematizing construction and maintenance of large networks by applying object oriented analysis, design and implementation techniques.

Based on work by Lauritzen and Nilsson [8] and Bangsø and Wuillemin [1] we present an Object Oriented modeling approach of influence diagrams with limited memory. Section 2 covers the fundamental definitions of these LImited Memory Influence Diagrams (LIMIDs), together with a solution-algorithm which exploits evaluation techniques of existing software. Following the framework of LIMIDs, Section 3 covers the definitions of a graphical object oriented LIMID specification language. The two theoretic pillars are illustrated by the slaughter pig example in Section 4.

2 Limited Memory Influence Diagrams

Motivated by the cause of being able to represent decision problems, where one for modeling or complexity reasons is unable to remember the entire past, [8] introduce the notion of LIMIDs. LIMIDs are specified similar to IDs, which are an extension of Bayesian Networks, see [5], with decisions and utility nodes. A LIMID is specified using a directed acyclic graph (DAG) with three types of nodes: chance (drawn as ellipses), decision (rectangular), or utility (diamond), see Figure 1(a). Contrary to IDs, no underlying assumption of "no-forgetting" exists in LIMIDs, i.e. every node available at the time of making a decision is explicitly specified by drawing an arc from this node into the decision node, see Figure 1(b). Every ID can be represented as a LIMID by explicitly drawing all no-forgetting arcs.

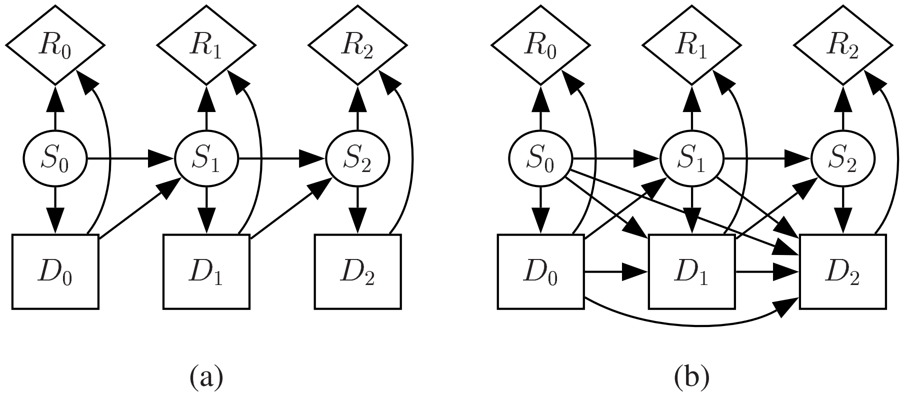


Figure 1: (a) A 3-slice Markov Decision process modeled as a LIMID. If the network in Figure (a) is interpreted as an ID instead, Figure (b) shows the corresponding LIMID, with all no-forgetting links made explicit.

We use a simple disease treatment decision scenario in the slaughter pen to illustrate the usability of LIMIDs to pig production. Simplicity is preferred over practical usability, thus our example lacks the direct applicability in real world pig production. A characteristic of slaughter pig production is that the pigs are identified on pen level rather than on individual level. Nevertheless our model deals with the treatment of a non-infectious disease in a single slaughter pig. Using the individual pig as level of abstraction our model illustrates an important point: No records exists for individual pigs, thus a veterinarian who has to decide whether to treat a slaughter pig cannot base his decision on previous information about the pig. This certainly violates the no-forgetting assumption of IDs, but fits perfectly into the framework of LIMIDs. Even in the case of access to all previous information, influence diagrams are not applicable, due to complexity problems in the generated cliques of the junction tree.

Whereas IDs can be solved explicitly as e.g. by [4], LIMIDs are solved by an iterative procedure called single policy updating. Given an initial strategy, Lauritzen and Nilsson [8] show, that single policy updating iteratively improves this initial strategy until it converges to a strategy, where no improvement of the expected utility can be obtained by changing the policy for at most one decision.

To make the LIMID-technology more explicit, we briefly describe it on a more formal level. Readers interested in more exhaustive description details are refered to [8]. Let $\mathcal{L}$ be a LIMID specified by a DAG $G = (V, E)$, where the set of nodes V is partitioned into disjoint sets of chance nodes Γ, decision nodes Δ, and utility nodes Υ. For $W \subseteq \Gamma \cup \Delta$ we denote by $\mathcal{X}_W$ the set of all possible configurations of the variables in W. If $d \in \Delta$, then a *pure policy*, denoted δ_d, is a function yielding a decision alternative in $\mathcal{X}_d$ for each configuration of $\mathrm{pa}(d)$, i.e.

$$\delta_d : \mathcal{X}_{\mathrm{pa}(d)} \times \mathcal{X}_d \to \{0,1\}, \quad \text{s. t. } \forall x_{\mathrm{pa}(d)} \in \mathcal{X}_{\mathrm{pa}(d)} : \sum_{x_d \in \mathcal{X}_d} \delta_d(x_{\mathrm{pa}(d)}, x_d) = 1.$$

Once a strategy $q = \{\delta_d : d \in \Delta\}$ has been specified, it induces a probability distribution of the variables in $\Gamma \cup \Delta$ determined by

$$f_q = \prod_{r \in \Gamma} p_r \prod_{d \in \Delta} \delta_d,$$

where p_r is the conditional probability function of r given its parents $\mathrm{pa}(r)$. Our goal is to find a strategy q that maximizes the expected total utility. That is, we try to maximize

$$\mathrm{EU}(q) = \sum_{x_{\Gamma \cup \Delta}} \left\{ f_q \left(\sum_{u \in \Upsilon} U_u \right) \right\}.$$

A *global maximum strategy* $\hat{q}$ is a strategy which satisfies that $\mathrm{EU}(\hat{q}) \geq \mathrm{EU}(q)$ for all strategies q. Let $q = \{\delta_d : d \in \Delta\}$ be a strategy and let $d_0 \in \Delta$. We define the *partially specified strategy* $q_{-d_0} = \{\delta_d : d \in \Delta \backslash \{d_0\}\}$. For any policy δ'_{d_0} for d_0 we denote the strategy obtained by replacing δ_0 with δ'_{d_0} in q as $\delta'_{d_0} * q$, in other words $\delta'_{d_0} * q = \{\delta'_{d_0}\} \cup q_{-d_0}$. This leads to two important definitions. Firstly, a *local maximum policy* for a strategy q at d, is a policy $\tilde{\delta}_d$ satisfying

$$\mathrm{EU}(\tilde{\delta}_d * q) = \sup_{\delta'_d} \mathrm{EU}(\delta'_d * q),$$

i.e. the best we can do at d given the policies of all other decisions stay fixed. Secondly, a *local maximum strategy*, $\tilde{q}$, is a strategy with all policies being local maximum policies. This means that we can not improve our strategy by changing at most one policy. Should we have found a local maximum strategy, it is not necessarily a global maximum strategy; a problem similar to that of local maxima when searching for a global maximum of a function.

If $d \in \Delta$, then a node $p \in \mathrm{pa}(d)$ is called a *non-requisite* parent of d if the utility function is not affected by removal of the information arc between p and d. Lauritzen and Nilsson [8] give an algorithm, which successively removes all graph deducible non-requisite arcs in $\mathcal{L}$. The resulting LIMID $\mathcal{L}_{\min}$ is called the *minimal reduction* of $\mathcal{L}$. Working on $\mathcal{L}_{\min}$ instead of $\mathcal{L}$ increases the efficiency of the solution algorithms, while yielding the same results with respect to strategy, expected utility, etc.

2.1 A LIMID Solution Algorithm

Lauritzen and Nilsson [8] describe a solution algorithm called *single policy updating*, which is guaranteed to converge towards a local maximum policy. One run of the algorithm consists of first arranging all $d \in \Delta$ in an evaluation order. Then following our evaluation order the local maximum policy for one d at the time is found, by keeping the other policies fixed and performing partial propagations. These propagations happen in a junction tree found by using ordinary Bayesian network moralization and triangulation techniques as in [5], ergo no strong triangulation is necessary. The problem is that this algorithm is not directly implementable using ordinary inference machines such as [3]. We thus propose the following "poor-man's" partial propagation algorithm, which uses sequential evaluation of influence diagrams with one decision.

Let $\mathcal{L}_{\min}$ be the minimal LIMID of a LIMID $\mathcal{L}$ and choose a solution order $d_1, \ldots, d_k$. It is often beneficial to let this order be concordant with the temporal order of the decisions in $\mathcal{L}$. Now, convert every d_i, $1 \leq i \leq k$, to a chance node. Repeat the following procedure until convergence of policy, or until the difference between to consecutive expected utility values is below some threshold.

◇ For $i = k, \ldots, 1$

- Convert d_i into a decision node, this results in an influence diagram I with one decision node, namely d_i. Compile I into a strong junction tree J according to the procedure of [4].
- Let C be the strong root of J with probability potential ϕ_C and utility potential ψ_C. Calculate $\gamma = \sum_{C \setminus \mathrm{fa}(d_i)} \phi_C \psi_C$.
- For each $x_{\mathrm{pa}(d_i)} \in \mathcal{X}_{\mathrm{pa}(d_i)}$, the best action is found as $x^*_{d_i} = \arg\max_{x_{d_i} \in \mathcal{X}_{d_i}} \gamma(x_{\mathrm{pa}(d_i)}, x_{d_i})$.
- Convert d_i to a chance node, and set $P(\mathcal{X}_{d_i} | \mathcal{X}_{\mathrm{pa}(d_i)})$ according to the pure policy found in the last step.

Our algorithm is slower than partial propagation, since more messages are sent. But each influence diagram we solve contains only one decision, hence the strong triangulation yields the same result as ordinary triangulation.

An important result of [8] is the identification of a class of so called *soluble* LIMIDs, possessing the ability that the above evaluation mechanism is guaranteed to find the global maximum strategy. It can be shown that the LIMID representation of any influence diagram, belongs to the class of soluble LIMIDs.

3 Object Oriented LIMIDs

Having introduced LIMIDs as a modeling tool for multi-stage decision scenarios with the same graphical specification techniques as Bayesian Networks (BNs), we also face the challenges with respect to development and maintenance of large networks. Recent research has focused on applying the abstraction and hierarchy concepts known from the Object Oriented paradigm to Bayesian Networks resulting in Object Oriented Bayesian Networks (OOBNs), see [1, 7]. The existence of the no-forgetting arcs in IDs has until now prevented all attempts to create a methodology for component based specifications of IDs. Since LIMIDs do not suffer from the yoke of no-forgetting arcs, component based modeling is possible. We show how to extend the graphical specifications language for OOBNs in [1] to LIMIDs, creating the notion of *Object Oriented LIMIDs* (OOLs). Due to space limitations we refrain from describing the class hierarchy notions of [1]; however it should be easy to adopt this to OOLs.

3.1 Class, Nodes, Restrictions

Following the work of Bangsø and Wuillemin [1] we introduce the following definitions. A *class* **C** is a DAG over (I, H, O) where I is a set of *input nodes*, H the set of *hidden nodes*, and O is the set of *output nodes*. $I \cup O$ is called the *interface* of **C**. This (I, H, O) partitioning of the nodes enables encapsulation of nodes into classes.

A node in **C** is either a *simple node* or an *instantiation* of another class. In OOP this corresponds to variables being either simple or abstract types. A simple node can either be a *reference node* or *real node*. Here a reference node is simply a place holder (an alias) for a simple node located somewhere else in the network, the so called *referenced node*. The OOP-analogue of reference nodes are reference variables. Real nodes are as in influence diagrams either chance, decision or utility nodes. By $\Gamma_{\mathbf{C}}, \Delta_{\mathbf{C}}, \Upsilon_{\mathbf{C}}$, we denote the set of simple chance, decision and utility nodes of a class **C**.

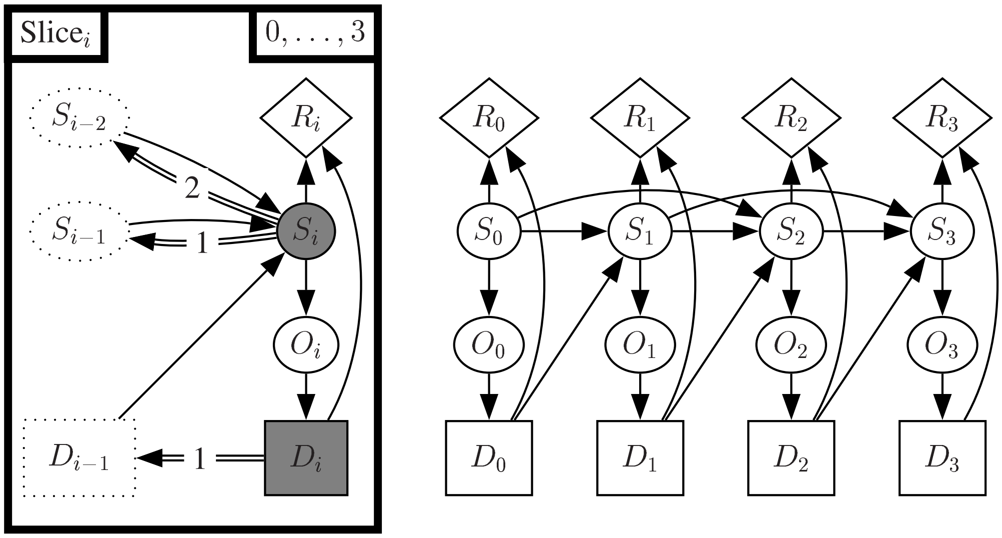
The links of the DAG come in two types. A *directed link* (Simple node $\rightarrow$ Real node) denotes, depending on the target node of the arrow, a probabilistic, information, or functional dependency. A *reference link* (Simple node $\Rightarrow$ Reference node) states that the target node is an alias for the source node, hence the two nodes will be merged into one node in the underlying network. A reference node can not have any incoming directed links and at most one incoming reference link. To avoid the specification of illegal LIMIDs, Bangsø and Wuillemin [1] do not allow reference links between two simple nodes belonging to the same class, and forbids a node to be referenced by more than one node in the same class.

A class **C** and each instantiation c of **C** must conform with the following semantic rules. The sets I, H, O are pairwise disjoint. Each node $i \in I$ has no parents in c, no children outside c, and at most one referenced node outside c. An $o \in O$ can not have parents outside c. To restrict the scope of each $h \in H$ to the body of c, h can not have children or parents outside c. Since utility nodes cannot have children, it makes no sense to allow them to be reference or referenced nodes, thus $(I \cup O) \cap \Upsilon_{\mathbf{C}} = \emptyset$. This last rule is the only one not directly following the OOBN specification rules of [1].

Performing an instantiation of the scope-wise outermost class of the network gives us the equivalent of a LIMID. It is possible to generate the *underlying LIMID* from an OOL by expanding all classes, collapsing reference nodes according to reference links, etc. This is necessary in order to use ordinary LIMID inference engines.

3.2 Repetitive Structures

Often problems in pig production are of a repetitive nature, where a fragment representing a time slice is continuously repeated with links between the slices plus a few auxiliary nodes and links. Naturally, this repeated fragment should be modeled using an OOL-class. To accommodate OOLs for repetitive structure we allow, following [1], cycles in the DAG — as long as one of the links in the cycle is a reference link. To further accommodate for use of past as found in time-series, we annotate the $\Rightarrow$ link with a positive integer, n,



(a) (b)

Figure 2: (a) A 2-step Partial Observed Markov Decision Process specified using a time slice OOL class with $\Rightarrow^n$ arcs. Dotted nodes denote import nodes, whereas filled nodes denote output nodes. The interval in the upper right corner denotes the number of slices wanted. (b) Shows the corresponding underlying LIMID.

such that $\Rightarrow^n$ has a meaning similar to the back-shift operator known from the time series literature. Figure 2(a) shows an application of the $\Rightarrow^n$ operator. Note also, that we do not explicitly have to show the import and export nodes of an instantiation when using it. It is sufficient to draw the nodes relevant for the specification.

Directed cycles of annotated reference links within a class are unfolded as follows. For slice number $i \in [\min, \ldots, \max]$, let t_i be the target of a $s_i \Rightarrow^n t_i$ link participating in a directed cycle.

- If $i - n < \min$, the link is ignored.
- If $\min \leq i - n \leq \max$, t_i references s_{i-n}.

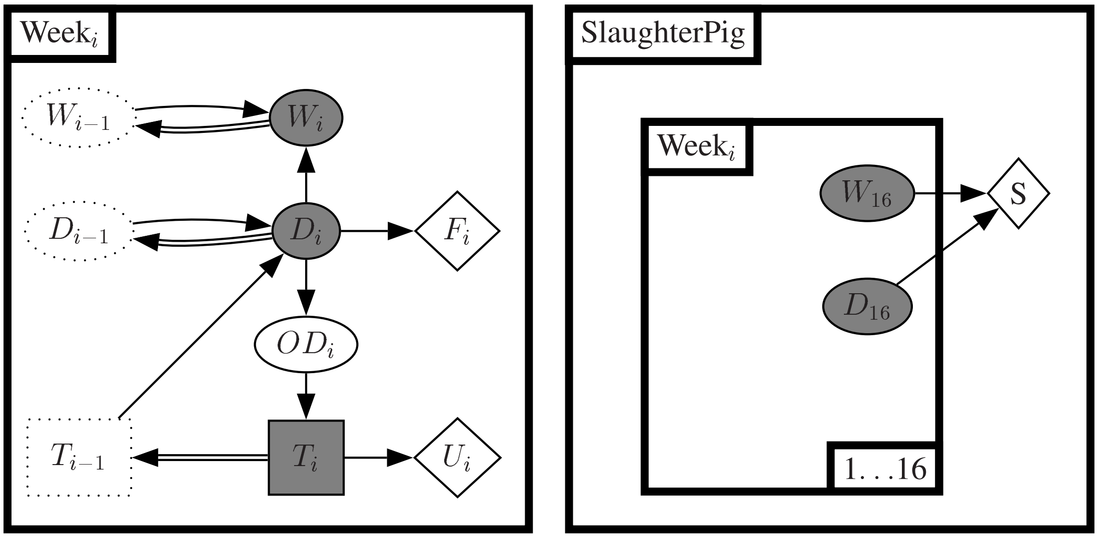
When generating the underlying network, reference nodes without referenced nodes will be deleted, while other reference nodes are merged as usual, see Figure 2(b).

4 Disease Treatment in the Slaughter Pen

Using the object-oriented approach of OOLs, we formulate a model describing a decision scenario, where a once-a-week visiting veterinarian has to decide whether to apply a specific treatment to a slaughter pig in order to cure a non-infectious disease influencing the growth rate.

4.1 Qualitative perspective

Using a top-down approach we begin our modeling by identifying different components in our model. Firstly, a component corresponding to the weekly decision scenario with treatment decisions influencing the disease state is needed. Secondly we need a component gathering the individual weeks and the reward received at slaughter time.



(a) The Week$_i$ class.

(b) The SlaughterPig class.

Figure 3: OOL specification of our model handling disease treatment in the slaughter pen. The network is created from two classes, Week$_i$ and SlaughterPig together forming a repetitive structure.

We formalize the decision scenario of the i'th week in a Week$_i$ class. As a first step, the nodes of the class are identified. Each week the veterinarian performs an insecure test to gain insight on presence of the disease in the slaughter pig. We assume, that the true disease state in the animal can be expressed as either *H*ealthy, *I*nfected, *R*ecovered, or *D*ead. To encourage application of treatment we assume that once the animal is recovered it is immune to further disease attacks. Based on the test result OD_i the decision T_i reflects whether the veterinarian decides whether to apply the treatment. The nodes are as follows.

Name	Description	States
W_i	A counter, summing the weeks spent sick.	$\{0, \dots, 16\}$
D_i	Disease stage	$\{H, I, R, D\}$
OD_i	Observed disease state through test	$\{healthy, diseased, dead\}$
T_i	Apply treatment?	$\{yes, no\}$

Figure 3(a) displays the Week$_i$ class in OOL notation. Notice, how the treatment of the last slice, i.e. T_{i-1}, influences the disease state D_i, and how the link from D_i to OD_i is used to specify the sensitivity and specificity of the applied test. Furthermore utility nodes U_i and F_i reflect the cost of treatment and the cost of feed, respectively.

It is asserted that the slaughter pig spends a total of 16 weeks in the slaughter pen before it is sent to the slaughterhouse. Hence, the SlaughterPig class shown in Figure 3(b) just contains 16 instantiations of the Week$_i$ class, linked together according to the reference links of Week$_i$. Sending the slaughter pig to the slaughterhouse, results in an income S, which depends on weight at delivery. We assume that the slaughter weight can be determined from the state of W_{16} and D_{16}.

4.2 Quantitative Perspective

After having specified the qualitative side of our network, we now turn to the specification of probability and utility functions. Due to space limitations only interesting parts of this

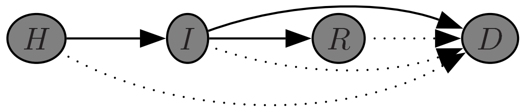


Figure 4: Disease model as CTMC. Note, the lack of transition from *Recovered* to *Healthy*. Dotted lines indicate transitions of the type any-state to death.

Table 1: Transition rate matrix $Q = [q_{ij}]$ for our CTMC.

T_{i-1}	No treatment				Treatment			
$D_{i-1} \backslash D_i$	H	I	R	D	H	I	R	D
H	-	0.0319	0.0000	0.0005	-	0.0151	0.0000	0.0005
I	0.0000	-	0.0151	0.0005	0.0000	-	0.0990	0.0005
R	0.0000	0.0000	-	0.0005	0.0000	0.0000	-	0.0005
D	0.0000	0.0000	0.0000	-	0.0000	0.0000	0.0000	-

specification are described in this section.

4.2.1 Probability specifications

Looking at the probability tables our OOL specification requires us to specify, we realize that the important part is to state a model for the disease development. Figure 3(a) reveals a Markov assumption, thus we choose to model the disease development by a continuous-time Markov Chain (CTMC), see [2]. Using a CTMC we gain the advantage of scalability with respect to time-step (e.g. days instead of weeks), and multiple transitions within a single time-step are handled naturally. The Markov chain will have the same states as D_i, its transition model is illustrated in Figure 4.

We assert constant transition probabilities, e.g. the death risk is independent of age, development of the disease in a healthy pig is independent on how long a pig has been healthy, etc. With the above assumptions it can be assumed that the transition probabilities of the Markov chain follow an exponential distribution. Based on the numbers in [8] we obtain a transition rate matrix as shown in Table 1.

To calculate the weekly transition probability matrix, $p_{ij}(7) = P(D_i = i | D_{i-1} = j)$, for both states of T_i we transform the transition rate matrix to a transition probability matrix, P for a small time interval, Δt. This interval is chosen sufficiently small, such that multiple transitions can be ignored. The element of P are thus calculated as,

$$p_{ij}(\Delta t) \approx q_{ij} \Delta t.$$

If $7/\Delta t$ is an integer we can use the Chapman-Kolmogorov Equations, see [2], to calculate the weekly transition matrix, i.e. $P^{(7)} = P^{7/\Delta t}$. In our case calculations show that $\Delta t = 7/256 = 0.0273$ is appropriate.

The probability table for each $P(W_i | D_i, W_{i-1})$ is just a sum construction, counting the number of D_j, $1 \leq j \leq i$, which are in state *Infected*. This number is used in the calculation of the slaughter utility function. Assuming our test treats both healthy and recovered animals as non-diseased, we let the sensitivity and specificity of the test in $P(OD_i | D_i)$ be 0.8 and 0.9 respectively. Dead animals are always diagnosed correctly.

4.2.2 Utilities

The cost of applying treatment, U_i is arbitrarily set to a price of DKK 10, feed consumption pr. week, F_i is set to $7 \cdot 2.5$ FU$_p$. Here, FU$_p$ is a Danish pig feed unit with $FU_p \approx 7.72$ MJ net energy at a price of 1.1 DKK pr.

Reward at slaughter time is zero if the pig died during the 16 weeks, otherwise each disease period results in a reduction in daily gain. For simplicity we assume a linear growth during the 16 weeks, hence any information about when the pig was sick is irrelevant,

$$S = r_w \cdot ko \cdot (W_0 + G' \cdot T) .$$

Here ko is the killing-out percentage, r_w is the price per kilo slaughter weight, W_0 is the weight at insertion, T is the total number of days in the slaughter pen, and G' is average daily gain corrected for presence of disease. We choose to calculate G' by assuming that the correct growth rate is linearly decreasing in number of weeks spent sick. That is,

$$G' = G - \frac{W_{16}}{16} R,$$

where G is the average daily gain of a disease free pig, and R is a reduction in daily gain due to presence of the disease. We select plausible values of $G = 800g$, $R = 30g$, and standard values for the remaining constants.

4.3 Solving the specified OOL

The LIMID solution algorithm from Section 2.1 was implemented in C++ using the Hugin API [3]. Running the program on the underlying LIMID of the OOL specified in this section results in convergence after three runs with the following local maximum strategy.

- For $T_1, \ldots, T_{12}$ apply the treatment if and only if the test is positive.
- For $T_{13}, \ldots, T_{16}$ no treatment is performed no matter the test result.

We do not know whether the found local maximum strategy actually is a global maximum strategy, but evaluating a large number random sample of the set of 6^{16} possible strategies has not revealed a better strategy.

Analyzing this strategy shows that in our model the cost of treatment in the late stages cannot be out-weighted by the increased growth of healthy pigs. In the early stages the situation is different, as the animal has a potential of staying sick for many weeks if not treated.

5 Conclusions and Future work

We now have a powerful object oriented specification language to design at least the qualitative part of a LIMID. To allow realistic OOP implementation of LIMIDs, a concept for integrating both qualitative and quantitative in e.g. a distinct OOL programming language is necessary. First then a consistent interaction between analysis, design and implementation by object oriented techniques exists.

Application of OOL modeling to our disease treatment problem revealed that treatment of a slaughter pig close to slaughter time might not be beneficial. We should although be careful not to put too much weight into conclusions from our rather constructed model. To make it more realistic, beneficial extensions would be to use the SlaughterPig class as building block to model an entire pen of slaughter pigs. Instead of modeling a virtual disease one should find an actual disease of relevance, e.g. the infectious swine influenza. The disease should be modeled using more disease states, while transition probabilities and the effect on growth could be based on actual data. A further necessity is to make the slaughter utility more coherent with the Danish slaughter pricing system.

LIMID solution algorithms face the same problems as hill-climbing techniques for maximizing of functions. It might be worth exploring hybrids between combinatoric optimization techniques and single policy updating, e.g. simulated annealing with local hill-climbing by single policy updating. Despite of the iterative nature of the solution algorithms, our tests show that convergence usually is achieved in only a handful of runs. Furthermore, promising techniques exists to speed up inference by exploiting the class structure during propagation. Thus, it should be straightforward to adopt the ideas of OOBNs and Multiple Sectioned Bayesian Network (MSBNs) in [1] to LIMIDs.

Acknowledgments

We thank Olav Bangsø, Aalborg University, for providing us with a preliminary version of [1].

References

[1] Bangsø, O. and P.-H. Wuillemin (2000). Object Oriented Bayesian Networks. A Framework for Topdown Specification of Large Bayesian Networks and Repetitive Structures. Technical report, Aalborg University, Hewlett-Packard Laboratory for Normative Systems. To appear.

[2] Hillier, F. S. and G. J. Lieberman (1990). *Introduction to Operations Research*. McGraw–Hill, fifth edition.

[3] Hugin Expert A/S (1998). *Hugin API 4.0 Reference Manual*. `http://www.hugin.dk/ftp/doc/api-manual.ps`.

[4] Jensen, F., F. V. Jensen and S. L. Dittmer (1994). From influence diagrams to junction trees. In de Mantaras, R. L. and D. Poole, eds., *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pp. 367–373. Morgan Kaufmann Publishers, San Francisco, CA, USA. ISBN 1-55860-3328.

[5] Jensen, F. V. (1996). *An introduction to Bayesian Networks*. UCL Press. ISBN 1-85728-332-5.

[6] Kaebling, L. P., M. L. Littman and A. R. Cassandra (1998). Planning and acting in partially observable stochastic domains. Artificial Intelligence, **101**(1–2): 99–134.

[7] Koller, D. and A. Pfeffer (1997). Object-oriented Bayesian networks. In Geiger, D. and P. P. Shenoy, eds., *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pp. 302–313. Morgan Kaufmann Publishers, San Francisco. ISBN 1-55860-485-5.

[8] Lauritzen, S. L. and D. Nilsson (2000). LIMIDs of Decision Problems. Submitted to Management Science.