Nilsson, Höhle:

Methods for evaluating Decision Problems with Limited Information

Sonderforschungsbereich 386, Paper 421 (2005)

Online unter: http://epub.ub.uni-muenchen.de/

Methods for evaluating Decision Problems with Limited Information

SFB386 Discussion Paper 421*

Dennis Nilsson
Codan Forsikring,
Business Intelligence Unit,
Denmark.

Michael Höhle
Department of Statistics,
University of Munich,
Germany.

Abstract

LImited Memory Influence Diagrams (LIMIDs) are general models of decision problems for representing limited memory policies (Lauritzen and Nilsson (2001)). The evaluation of LIMIDs can be done by *Single Policy Updating* that produces a local maximum strategy in which no single policy modification can increase the expected utility. This paper examines the quality of the obtained local maximum strategy and proposes three different methods for evaluating LIMIDs. The first algorithm, Temporal Policy Updating, resembles Single Policy Updating. The second algorithm, Greedy Search, successively updates the policy that gives the highest expected utility improvement. The final algorithm, Simulating Annealing, differs from the two preceeding by allowing the search to take some downhill steps to escape a local maximum. A careful comparison of the algorithms is provided both in terms of the quality of the obtained strategies, and in terms of implementation of the algorithms including some considerations of the computational complexity.

1 Introduction

LImited Memory Influence Diagrams or simply LIMIDs are general models of decision problems for representing limited memory policies (Lauritzen and Nilsson (2001)). In contrast to traditional influence diagrams (IDs), LIMIDs have no assumption of no-forgetting, and consequently the LIMID framework is an extension of IDs. The study of such policies is motivated in a number of ways, as detailed below.

Limited memory policies are not only simpler; they may also be optimal: In many cases previous events can be ignored because they are irrelevant given all other available information. That is indeed the case in the class of Markovian environments, in which the optimal choice of a particular decision is completely determined by the latest observation leaving all other observations irrelevant. Similarly, in IDs it is well-known that optimal decisions can be made by only considering the so-called *requisite information* observed up until the

*This paper is also available electronically from `http://www.statistik.lmu.de/sfb386/`

decisions are made (Shachter 1998; Nielsen and Jensen 1999; Nilsson and Lauritzen 2000; Jensen 2001).

'Full memory policies' can also be computationally expensive. This holds, for example, for partially observable Markov decision problems, also known as POMDPs (Lovejoy (1991), Kaelbling *et al.* (1998)). Here, optimal policies are based on the whole history, i.e. all previous observations and all previous actions. Even though there exists exact algorithms for solving POMDPs, general POMDPs are intractable. Similarly, the performance of IDs in practice has been disappointing because the existing algorithms compute policies from all requisite information; a procedure that is bound to fail for most real world applications.

When leaving the territory of 'full memory policies', finding global optimal policies may be intractable. Lauritzen and Nilsson (2001) present a LIMID algorithm, termed *Single Policy Updating*, which produces a local maximum strategy in which no single policy modification can increase the expected utility. The algorithm is exact in some, 'soluble', cases, and only approximate in other, 'non-soluble', cases.

Despite the large number of potential applications of non-soluble LIMIDs, we only have little knowledge on the quality of the local maximum strategies produced by Single Policy Updating. This paper investigates three algorithms for evaluating LIMIDs in the non-soluble case. The first algorithm, Temporal Policy Updating, resembles Single Policy Updating. The second algorithm, Greedy Search, successively updates the policy that gives the highest expected utility improvement. The final algorithm, Simulating Annealing, differs from the two preceeding by allowing the search to take some downhill steps to escape a local maximum. Instead of picking the best move, it picks a random move. The random move dictates to make an inferior decision with a low, but positive probability. A careful comparison of the algorithms is provided both in terms of the quality of the obtained strategies, and in terms of implementation of the algorithms including some considerations of the computational complexity.

We will illustrate our methods on a robot navigation problem used by e.g. Littman *et al.* (1995) and Horsch and Poole (1998). The quantitative specifications are provided from the latter source.

The Maze Problem

An agent is traversing a maze to reach a goal state. To begin with, the agent is placed at random in one of 22 valid non-goals and the task is to navigate to the goal state in at most 10 steps. The maze is shown in Figure 1.

Four sensors, one in each compass direction inform the agent about presence of walls in neighbouring tiles. Based on this information, the agent decides whether to move n, s, e, w, or stay in its current position. After 10 steps the game ends and a reward of 1 is given if the current state is the goal state, otherwise no reward is given.

Four agents are examined with different ability to act and sense. The sensor of an agent can either be perfect or noisy and the actuators of an agent can also either be perfect or noisy. With perfect actors, moving in a direction where there is a wall always fails. Otherwise the desired moves always succeeds. Noisy actors on the other hand have a 0.089 probability of not moving, a 0.001 probability of moving in the opposite direction, a probability of 0.01 for moving in a +90 degree direction and finally a 0.01 probability for the -90 direction. Except for stay actions which always succeed, the probability of moving in the desired direction is thus 0.89. In case of walls, all directions containing a wall are assigned zero probability and the remaining probabilities are normalized. With perfect sensors, a wall is detected if and only if there is a wall. With noisy sensors, a

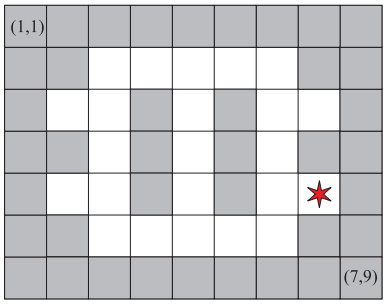

Figure 1: Maze environment. Shaded tiles indicate walls and the star denotes the goal state.

sensor fails to recognize a wall with probability 0.1, and with probability 0.05 a wall is reported even though no existed.

Whereas such decision problems are typically represented as POMDPs, our model uses the LIMID representation. The LIMID framework has no assumption of no-forgetting and consequently it can easily model the situation in which the agent does not remember previous observations and previous actions. This ability is crucial, because the magnitude of all previous observations (sensor inputs) grows exponentially and makes it prohibitive to find an optimal strategy.

2 Limited Memory Influence Diagrams

The present section gives a basic description of LIMIDs as developed in Lauritzen and Nilsson (2001). For proofs, the reader is referred to this source.

2.1 LIMID Representation

A LIMID is a directed acyclic graph (DAG) with no cycles. There are three types of nodes in the graph: *Chance nodes*, shown as circles, represent random variables. *Decision nodes*, shown as boxes, represent decision variables. Finally, *value nodes*, shown as diamonds, represent (local) utility functions.

We are given a LIMID $\mathcal{L}$ with chance nodes Γ, decision nodes Δ, and utility nodes Υ. We let $V = \Gamma \cup \Delta$. Elements in V will be termed *nodes* or *variables* interchangeably. The variable $v \in V$ is taking values in a discrete space $\mathcal{X}_v$. For $A \subseteq V$ we write $\mathcal{X}_A$ for $\times_{v \in A} \mathcal{X}_v$. Typically elements of $\mathcal{X}_A$ are called *configurations* and denoted by $x_A = (x_v : v \in A)$. The set of *parents* of a node n is denoted $\mathrm{pa}(n)$, and the *family* of d is given by $\mathrm{fa}(d) = \mathrm{pa}(d) \cup \{d\}$. For $S \subseteq V$, we let $\mathrm{fa}(S)$ be a short-hand notation for $\cup\{\mathrm{fa}(n) : n \in S\}$. Further, $\mathrm{de}(n)$ denotes the descendants of node n.

Arcs in a LIMID have a different meaning depending on which node they go into. Arcs into chance nodes denote probabilistic (or functional) dependence. Associated with each chance node r is a nonnegative real function p_r on $\mathcal{X}_r \times \mathcal{X}_{\mathrm{pa}(r)}$ such that for each configuration of $\mathcal{X}_{\mathrm{pa}(r)}$, p_r adds to 1 when summing over the possible configurations in $\mathcal{X}_r$. The arcs into value nodes are functional. Associated with each value node u, there is a utility function U_u on $\mathcal{X}_{\mathrm{pa}(u)}$. Arcs into a decision node are informational since they originate from nodes whose values are known for the decision maker when the decision is to be made. In contrast with

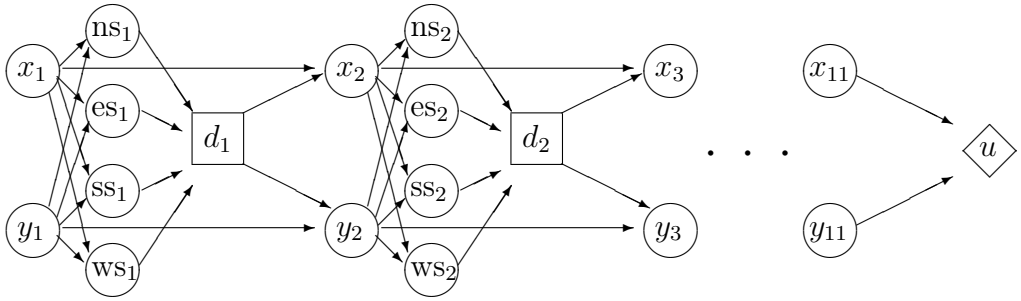

Figure 2: Basic structure of the LIMID representing the maze example. All informational arcs are explicitly depicted.

traditional IDs, LIMIDs have no assumption of 'no forgetting'[1].

Figure 2 shows a LIMID representation of the maze problem. The nodes x_i and y_i denote the horizontal and vertical position of the agent in the grid at time i. In our representation of the problem, the only informational arcs into decision node d_i originate from the current sensor inputs $\{ns_i, es_i, ss_i, ws_i\}$. Consequently, our model represents the decision problem where the agent only remembers the current sensor inputs when decision d_i is to be made. A more compact representation would be possible using an object oriented framework (Koller and Pfeffer (1997), Bangsø and Wuillemin (2000), Höhle *et al.* (2000)).

2.2 Policies and strategies

A *policy* for $d \in \Delta$ is a nonnegative real function δ_d on $\mathcal{X}_d \times \mathcal{X}_{\mathrm{pa}(d)}$ such that for each configuration of $\mathcal{X}_{\mathrm{pa}(d)}$, δ_d adds to 1 when summing over the possible configurations in $\mathcal{X}_d$. A policy for d is said to be *pure*, if it prescribes a unique alternative of $\mathcal{X}_d$ for each possible configuration $x_{\mathrm{pa}(d)} \in \mathcal{X}_{\mathrm{pa}(d)}$. Further, a policy is said to be *uniform*, and denoted $\bar{\delta}_d$, if $\bar{\delta}_d \equiv 1/|\mathcal{X}_d|$.

A *strategy* $q = \{\delta_d : d \in \Delta\}$ determines a joint distribution of all the variables in V as (abbreviating x_V and $\mathcal{X}_V$ to x and $\mathcal{X}$ respectively)

$$f_q(x) = \prod_{r \in \Gamma} p_r(x_r | x_{\mathrm{pa}(r)}) \prod_{d \in \Delta} \delta_d(x_d | x_{\mathrm{pa}(d)}), \quad x \in \mathcal{X}.$$

A strategy is *uniform* if it consists of uniform policies only. Similarly, a strategy is *pure* if it consists of pure policies only. If a strategy is not pure it is called *random*.

The expected utility of q is the expectation of the total utility $U = \sum_{u \in \Upsilon} U_u$ wrt. the joint distribution of V induced by q:

$$\mathrm{EU}(q) = \sum_{x \in \mathcal{X}} f_q(x) U(x).$$

Throughout we shall use the notion $q * \tilde{\delta}_d$, where $q = \{\delta_d : d \in \Delta\}$ is a strategy and $\tilde{\delta}_d$ is a policy for $d \in \Delta$, to denote the strategy $q' := q \setminus \{\delta_d\} \cup \{\tilde{delta}_d\}$. So, q' is obtained from q by replacing δ_d by $\tilde{\delta}_d$.

[1]The assumption of 'no forgetting' implies that decision maker does not forget: If the value of a variable is known to the decision maker when a decision is to be made, then the variable is also known at the time of all subsequent decisions.

A *local maximum policy* for a strategy $q = \{\delta_d : d \in \Delta\}$ at d is a policy $\tilde{\delta}_d$ that satisfies

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

where the supremum is taking over the set of possible policies for d. Further, a *local maximum strategy* has the property that all its policies are local maximum policies, i.e., for all $d \in \Delta$ and all policies δ_d we have that $\mathrm{EU}(q) \geq \mathrm{EU}(q * \delta_d)$. Finally, a *global maximum strategy* $\hat{q}$ is a strategy that maximizes the expected utility, i.e. $\mathrm{EU}(\hat{q}) \geq \mathrm{EU}(q)$ for all strategies q.

2.3 Single Policy Updating

Single Policy Updating (SPU) is an iterative procedure for evaluating general LIMIDs. It starts with the uniform strategy and proceeds by modifying (updating) the policies in a random or systematically order. If the current strategy is $q = \{\delta_d : d \in \Delta\}$ and the policy for $d_i \in \Delta$ is to be updated, then SPU essentially consists of two steps:

Optimize: Compute a new policy for d_i:

$$\tilde{\delta}_{d_i} = \arg\sup_{\delta_{d_i}} \mathrm{EU}\left(q * \delta_{d_i}\right),$$

where the supremum is taking over the set of possible policies.

Replace: Redefine $q := q * \tilde{\delta}_{d_i}$.

So, SPU computes a local maximum policy $\tilde{\delta}_{d_i}$ for q at d_i, and replaces the current policy for d_i by $\tilde{\delta}_{d_i}$. During SPU, the policies are updated until they converge to a local maximum strategy where no single policy modification can increase the expected utility.

SPU is an iterative improvement algorithm, i.e. after each policy updating, the expected utility of the current strategy has increased or is unaltered. Further, SPU converges to a local maximum strategy in a finite number of updating steps. Convergence is reached, if no single policy updating can increase the expected utility. Section 2.5 describes how SPU can be performed by a simple message passing algorithm in a junction tree.

2.4 Potentials

During evaluation of the decision problem, the quantative elements of our LIMID is represented through entities called potentials.

Let $W \subseteq V$. A *potential* on W consists of a pair of real-valued functions $\phi_W = (p_W, u_W)$ on $\mathcal{X}_W$. The first part p_W is called the *probability part* and is non-negative, and the second part is called the *utility part*. We shall need two operations on potentials.

Definition 1 Let $\phi_{W_1} = (p_{W_1}, u_{W_1})$ and $\phi_{W_2} = (p_{W_2}, u_{W_2})$ be two potentials on W_1 and W_2. The *combination* of ϕ_{W_1} and ϕ_{W_2} is defined by

$$\phi_{W_1} \otimes \phi_{W_2} = (p_{W_1} p_{W_2}, u_{W_1} + u_{W_2}).$$

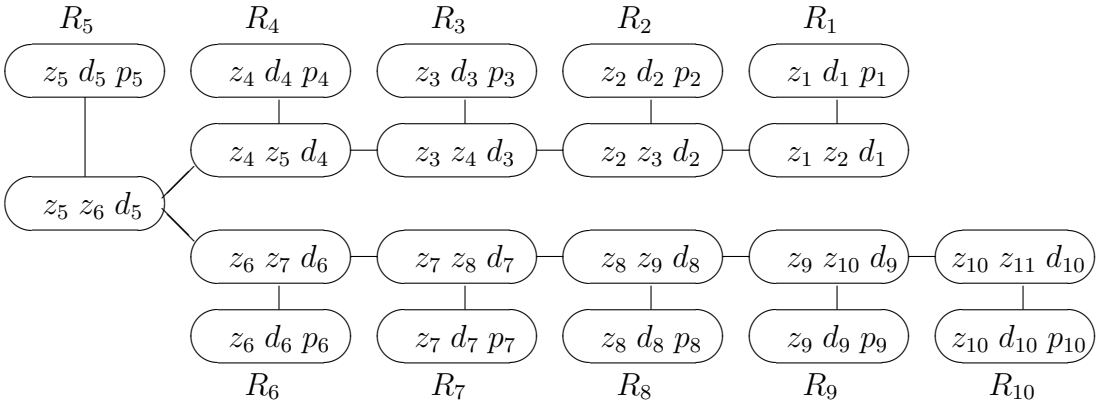

Figure 3: Junction tree for the maze problem. A short-hand notation has been used: $p_i = \{ns_i, es_i, ss_i, ws_i\}$, $z_i = \{y_i, x_i\}$. Clique R_i contains decision d_i and its parents $\mathrm{pa}(d_i) = p_i$.

Definition 2 Let $\phi_W = (p_W, u_W)$ be a potential on W, and let $W_1 \subseteq W$. The *marginalization* $\phi_W^{\downarrow W_1}$ of ϕ_W on W_1 is given by

$$\phi_W^{\downarrow W_1} = \left(\sum_{W \setminus W_1} p_W, \frac{\sum_{W \setminus W_1} p_W u_W}{\sum_{W \setminus W_1} p_W} \right),$$

where $\sum_{W \setminus W_1}$ is defined as the 'usual' marginal and $\frac{a}{0} = 0$ for all a.

The notion of potential is similar to what is used in Shenoy (1992), Jensen *et al.* (1994), and Cowell *et al.* (1999). Further, the above definitions of combination and marginalization satisfy the Shenoy-Shafer axioms (Shenoy and Shafer (1990)).

2.5 Junction Tree Representation

The whole decision problem can be represented and evaluated in terms of a junction tree.

A process of compilation, involving various graph-manipulations, such as moralization and triangulation is performed on the LIMID to make it amenable for evaluation. The result of the compilation is a tree of cliques $\mathcal{C}$, the *junction tree*. Figure 3 presents the junction tree for the LIMID representation of the maze problem.

After initializing the junction tree, a potential ϕ_C is associated with each clique C. The *joint potential* is defined by the combination of all the cliques potentials:

$$\phi_V = \otimes_{C \in \mathcal{C}} \phi_C = \left(\prod_{r \in \Gamma} p_r \prod_{d \in \Delta} \delta_d, \sum_{u \in \Upsilon} U_u \right).$$

SPU evaluates the decision problem by message passing in the junction tree. The structure of a *message* from clique C_1 to clique C_2 is given by

$$\phi_{C_1 \to C_2} = (\phi_{C_1} \otimes (\otimes_{C \in \mathrm{ne}(C_1) \setminus C_2} \phi_{C \to C_1}))^{\downarrow C_2}, \tag{1}$$

where $\mathrm{ne}(C_1)$ are the neighbours of C_1 in the junction tree, and $\phi_{C \to C_1}$ is the message from C to C_1. The message passing in (1) corresponds to the Shafer-Shenoy propagation scheme described in Shafer and Shenoy (1990).

1. **Step 1:** Retract the policy for d from the potential on R to obtain $\tilde{\phi}_R$.
2. **Step 2:** Collect messages to R to obtain ϕ^*_R.
3. **Step 3:** Compute $\phi^*_{\mathrm{fa}(d)} = (\phi^*_R)^{\downarrow \mathrm{fa}(d)}$.
4. **Step 4:** Compute the contraction $c_{\mathrm{fa}(d)}$ of $\phi^*_{\mathrm{fa}(d)}$.
5. **Step 5:** For each $x_{\mathrm{pa}(d)}$, find a configuration $x^*_d \in \mathcal{X}_d$ satisfying

$$x^*_d = \arg\max_{x_d} c_{\mathrm{fa}(d)}(x_d, x_{\mathrm{pa}(d)}),$$

   and define $\tilde{\delta}_d(x_d, x_{\mathrm{pa}(d)})$ as 1 if $x_d = x^*_d$ and 0 otherwise.
6. **Step 6:** Add $\tilde{\delta}_d$ to the potential on R.

Table 1: Single Policy Updating (SPU). The updating of the policy for decision d, where d is assigned to root-clique R.

Let $\phi_W = (p_W, u_W)$ be a potential on W. The *contraction* of ϕ_W, denoted $\mathrm{cont}(\phi_W)$, is defined as the real-valued function on $\mathcal{X}_W$ given as $\mathrm{cont}(\phi_W) = p_W u_W$.

Table 1 shows the computational steps performed by SPU. Note that all steps, except the collect step (Step 2), consists of computations within the root-clique R.

2.6 Soluble LIMIDs

This subsection provides a characterization of LIMIDs for which a global maximum strategy is achieved during Single Policy Updating.

An *optimum policy* $\hat{\delta}_d$ for decision d is a policy that maximizes the expected utility among all policies for d whatever the other policies in $\mathcal{L}$ are. In other words, an optimum policy is a local maximum policy for all strategies q in $\mathcal{L}$. It is not all decision nodes that have an optimum policy, but below we provide a graphical characterization of decision nodes that has an optimum policy. For this purpose, we apply the notion of *irrelevance* as expressed through d-separation (Pearl (1986)). Let the symbolic expression

$$A \perp_{\mathcal{L}} B \mid S$$

denote that A and B are d-separated by S in the DAG formed by all the nodes in the LIMID $\mathcal{L}$, i.e. including the utility nodes.

A decision node d_0 in $\mathcal{L}$ having decision nodes Δ, is said to be *extremal* in $\mathcal{L}$, if

$$u \perp_{\mathcal{L}} \mathrm{fa}(\Delta \setminus \{d_0\}) \mid \mathrm{fa}(d_0) \tag{2}$$

for every utility node $u \in \mathrm{de}(d_0) \cap \Upsilon$. It can be shown that if d_0 is extremal in $\mathcal{L}$, then it has an optimum policy. Further, the optimum policy for d_0 is simply a local maximum policy at d_0 for the uniform strategy. The latter provides a simple method for computing an optimum policy: Start with the uniform strategy in $\mathcal{L}$ and update the policy for d_0 using SPU (see Table 1).

A LIMID $\mathcal{L}$ is said to have an *exact solution ordering* $d_1, \ldots, d_k$, if for all i, d_i is extremal in $\mathcal{L}$ when $d_{i+1}, \ldots, d_k$ have been converted into chance nodes. In this case, $\mathcal{L}$ is said to be *soluble*, and it can be shown that SPU converges to a global maximum strategy when the policies initially are uniform, and they are updated using the order $d_k, \ldots, d_1$.

2.7 Redundant information

Redundant information in a LIMID $\mathcal{L}$ is expressed by informational arcs that can be ignored because they are irrelevant. So, irrelevant informational arcs can be removed from $\mathcal{L}$ without effecting the global maximum strategy for $\mathcal{L}$. The results can be used to reduce the complexity of evaluating $\mathcal{L}$ during SPU.

A node $n \in \mathrm{pa}(d)$ in $\mathcal{L}$ is said to be *non-requisite* for d, if

$$n \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d)) \mid (\mathrm{fa}(d) \setminus \{n\}),$$

and in this case we say that the informational arc from n to d is non-requisite. A node (arc) that is not non-requisite is said to be *requisite*. For instance, it can be seen that all informational arcs in the LIMID in Figure 2 are requisite.

A *reduction* of $\mathcal{L}$ is obtained by successively removing non-requisite informational arcs in $\mathcal{L}$. A *minimal reduction* of $\mathcal{L}$, denoted $\mathcal{L}_{\min}$, is a reduction of $\mathcal{L}$ in which all informational arcs are requisite. Every LIMID has a unique minimal reduction.

As shown in Lauritzen and Nilsson (2001), extremality is preserved under reduction, i.e. if d is extremal in $\mathcal{L}$, then d is extremal in any reduction $\mathcal{L}'$ of $\mathcal{L}$. Further, solubility and the maximum expected utility is also preserved under reduction. In particular these properties lead to more efficient algorithms for solving influence diagrams (Nilsson and Lauritzen (2000), and Madsen and Nilsson (2001)).

3 Evaluating non-soluble LIMIDs

The search for a global maximum strategy in a non-soluble LIMID $\mathcal{L}$ is computational prohibitive in general. So, we must rely on approximate methods for the evaluation of $\mathcal{L}$. This section proposes three algorithms for this task.

The first algorithm, Temporal Policy Updating resembles SPU. The second algorithm, Greedy Search, successively updates the policy that gives the highest expected utility improvement. The final algorithm, Simulating Annealing, differs from the two preceeding algorithms by dictating an inferior decision with a low, but positive probability.

For computational efficiency, we assume that our starting point is a LIMID $\mathcal{L}$ where non-requisite arcs have been removed – e.g. by the use of the methods in 2.7. Further, we assume that $\mathcal{L}$ has no extremal decision nodes since, as explained in Section 2.6, the case with one extremal decision node d is easily handled by an initial computation of an optimum policy $\tilde{\delta}_d$, and then implement $\tilde{\delta}_d$ by converting d into a chance node with $\tilde{\delta}_d$ as the associated conditional probability distribution.

3.1 Temporal Policy Updating

It is wellknown that for soluble LIMIDs, the exact solution order is the optimal policy updating order (Section 2.6). For non-soluble LIMIDs however, we have no knowledge of a superior updating order. In this subsection, we will argue that it is sensible to use a temporal solution order:

Definition 3 A *temporal solution ordering* $(d_1, \ldots, d_n)$ of the decision nodes in $\mathcal{L}$, is an ordering with the property that for all i, there is no directed path from d_i to any nodes in $\{d_{i+1}, \ldots, d_n\}$.

The acyclicity of LIMIDs ensures the existence of a temporal solution order. For instance, the LIMID in Figure 2 has a unique temporal solution order $(d_1, \ldots, d_{10})$.

For soluble LIMIDs, temporal solution orders and exact solution orders coincide in the following sense:

Theorem 1 *If $\mathcal{L}$ is soluble, then there exists an exact solution order of $\mathcal{L}$, which also is a temporal solution order of $\mathcal{L}$.*

To prove Theorem 1, we use that d-separation obeys the *graphoid axioms* (Verma and Pearl (1990)). In particular we will need the following axiom: For any subsets A, B, C, and D of nodes in $\mathcal{L}$ we have that

$$\text{if } A \perp_{\mathcal{L}} B \mid C \text{ and } D \subseteq B \text{ then } A \perp_{\mathcal{L}} D \mid C. \tag{3}$$

Proof (Theorem1): Suppose $(d_1, \ldots, d_n)$ is an exact solution order in $\mathcal{L}$, and define T as the set of pairs given by

$$T \;=\; \{(i,j) \mid i < j,\ d_i \in \mathrm{de}(d_j)\}. \tag{4}$$

We want to show that $(d_1, \ldots, d_n)$ is a temporal solution order, i.e. $T = \emptyset$. We use the shorthand notation $\Delta_j = \{d_1, \ldots, d_j\}$, and $\Upsilon_j = \mathrm{de}(d_j) \cap \Upsilon$. Without loss of generality, we assume that $\Upsilon_j \neq 0$ for all j. The proof is by contradiction.

Suppose $(a, b) \in T$. Since $(d_1, \ldots, d_n)$ is an exact solution ordering we have

$$\Upsilon_b \perp_{\mathcal{L}} \mathrm{fa}(\Delta_{b-1}) \mid \mathrm{fa}(d_b). \tag{5}$$

Because $(a, b) \in T$, it follows that $\Upsilon_a \subseteq \Upsilon_b$, and $\mathrm{fa}(\Delta_a) \subseteq \mathrm{fa}(\Delta_{b-1})$. So, from (5) and the graphoid axiom (3) we obtain

$$\Upsilon_a \perp_{\mathcal{L}} \mathrm{fa}(\Delta_a) \mid \mathrm{fa}(d_b). \tag{6}$$

Since $\Upsilon_a \subseteq \mathrm{de}(d_a)$ and $d_a \in \mathrm{de}(d_b)$, there is a directed path from d_a to Υ_a containing no nodes from $\mathrm{fa}(d_b)$, i.e. $\Upsilon_a \not\perp_{\mathcal{L}} d_a \mid \mathrm{fa}(d_b)$. Thus $\Upsilon_a = \emptyset$, and we have reached a contradiction. The result follows. □

Theorem 1 provides a heuristic argument for using a temporal solution order for evaluating non-soluble LIMIDs: Typically, a non-soluble LIMID $\mathcal{L}'$ has emerged by removing informational arcs from a soluble LIMID $\mathcal{L}$ having exact solution ordering, say $(d_1, \ldots, d_n)$. It seems sensible to update the policies in $\mathcal{L}'$ using the same ordering $(d_1, \ldots, d_n)$ because this ordering is optimal for solving $\mathcal{L}$. A natural question rises: Can we deduce an exact solution ordering of $\mathcal{L}$ from $\mathcal{L}'$? The following corollary gives an answer to this question:

Corollary 1 *Suppose $\mathcal{L}'$ is obtained by removing informational arcs from a soluble LIMID $\mathcal{L}$. If $(d_1, \ldots, d_n)$ is a unique temporal solution order in $\mathcal{L}'$, then $(d_1, \ldots, d_n)$ is an exact solution order in $\mathcal{L}$.*

Proof: Suppose $\mathcal{L}'$ is obtained by removing informational arcs from $\mathcal{L}$, and assume $(d_1, \ldots, d_n)$ is a unique temporal solution order in $\mathcal{L}'$. Then, $(d_1, \ldots, d_n)$ is also a unique temporal solution order in $\mathcal{L}$ and, by Theorem 1, $(d_1, \ldots, d_n)$ is an exact solution order in $\mathcal{L}$. □

The conditions in Corollary 1 are fulfilled for the Maze problem since the associated LIMID (Figure 2) has a unique temporal solution ordering $(d_1, \ldots, d_{10})$. So, by Corollary 1,

Input:

- A minimal reduction $\mathcal{L}$ with temporal solution order $(d_1, \ldots, d_n)$;
- Uniform strategy $q_0 := \{\bar{\delta}_d : d \in \Delta\}$ and its expected utility $EU(q_0)$;
- $\epsilon > 0$;

for $a := 1$ **to** ∞ **do**

1. $q_a := q_{a-1}$;
2. for $j := n$ to 1 do
   (a) **apply** SPU (see Table 1) for decision d_j to obtain $\tilde{\delta}_{d_j}$;
   (b) **let** $q_a := q_a * \tilde{\delta}_{d_j}$ and **compute** expected utility $EU(q_a)$;
3. **if** $EU(q_a) - EU(q_{a-1}) < \epsilon$ **then stop**.

Table 2: The pseudo-code for Temporal Policy Updating.

$(d_1, \ldots, d_{10})$ is also an exact solution ordering for the original (soluble) version of the problem containing all no-forgetting arcs. It is easily seen that this is indeed the case.

After introducing temporal solution orders, we now present the algorithm Temporal Policy Updating for evaluating a non-soluble LIMID $\mathcal{L}$. The pseudo-code is given in Table 2. The algorithm is essentially SPU using a temporal solution order, say $(d_1, ..., d_n)$, of $\mathcal{L}$. Temporal Policy Updating starts from a uniform strategy and initially computes a local maximum policy for d_n. Then, a local maximum policy for d_{n-1} is computed and so forth. The computations are performed by the propagation of flows in a junction tree $\mathcal{T}$. If d_j $(j = 1, \ldots, n)$ is assigned to root-clique $R_j \supseteq fa(d_j)$ in $\mathcal{T}$, then the initial policy updating of d_n is done by collecting flows towards R_n. Similarly, the policy updating of d_{n-1} may be computed by collecting flows towards R_{n-1}. For efficiency reasons, however, we may only perform a partial collect, in which flows are only passed along the unique path from R_n towards R_{n-1}. Likewise, when subsequent policies are updated, a partial collect propagation is sufficient. Details can be found in Lauritzen and Nilsson (2001).

3.2 Greedy Search

Greedy Search updates the policy that gives the highest increase in expected utility. So, in each updating step, Greedy Search examines the result of updating each policy, and updates the policy that gives the highest increase in expected utility. Typically, this algorithm demands fewer policy updates before convergence, but each updating step is more time-consuming than in the previous algorithm.

A pseudo-code of Greedy Search is presented in Table 3. During each policy updating it is advantageous to distribute flows as follows: Suppose d_j is assigned to root-clique R_j, and assume the policy for d_j is the last policy that has been updated. The next policy to be updated is the policy that gives the highest increase in expected utility (i.e. step 1(a)-(b) in the algorithm). In principle, we could search for this policy by performing $n - 1$ collect propagations towards each of the other root-cliques R_k, $k \neq j$. This usually involves a great deal of duplication. Instead, we distribute flows from R_j towards the other root-cliques. After this 'distribution-phase', the increase in expected utility by updating the policy for d_k, $k \neq j$ can be computed by with-in clique computations of the root-clique R_k.

Input:

- A minimal reduction $\mathcal{L}$ with decision nodes $\Delta = \{d_1, ..., d_n\}$;
- Uniform strategy $q_0 := \{\bar{\delta}_d : d \in \Delta\}$ and its expected utility $EU(q_0)$;
- $\epsilon > 0$;

for $a := 1$ **to** ∞ **do**

1. **for** $j := 1$ **to** n **do**
   (a) **perform** Step 1–5 in SPU (see Table 1) for decision d_j to obtain $\tilde{\delta}_{d_j}$;
   (b) **compute** expected utility $EU(q_{a-1} * \tilde{\delta}_{d_j})$;
2. **let** $j' := \arg\max_j EU(q_{a-1} * \tilde{\delta}_{d_j})$ and **do**
   (a) **add** $\tilde{\delta}_{d_{j'}}$ to the potential of the root-clique of $d_{j'}$;
   (b) for every $j \neq j'$, **re-add** the retracted policy for d_j to the potential of the root-clique of d_j;
   (c) **let** $q_a := q_{a-1} * \tilde{\delta}_{d_{j'}}$;
3. **if** $EU(q_a) - EU(q_{a-1}) < \epsilon$ **then stop**.

Table 3: The pseudo-code for Greedy Search.

3.3 Simulating Annealing

The problem with the two preceeding algorithms is that both algorithms may quickly get stuck on a local maximum. Simulated Annealing, on the other hand, allows the search to take some downhill steps to escape the local maximum. Instead of picking the best move, it picks a random move. The algorithm selects a randomized strategy for the agent, such that the best move is chosen with a probability less than one, and the remaining (bad) moves are chosen with equal probability. Because random strategies are handled and computed efficiently in LIMIDs, our implementation of Simulated Annealing is obtained by a simple and efficient modification of SPU.

Table 4 shows the pseudo-code of Simulated Annealing algorithm. A temperature function is used to determine the probability of picking the best move. At higher 'temperatures', 'bad' moves are more likely. Suppose decision d is to be updated and x_d^* is an optimal move whereas $x_d^{'}$ is non-optimal. Then the updated policy δ_d is determined by

$$\frac{\delta_d(x_d^{'} \mid x_{\mathrm{pa}(d)})}{\delta_d(x_d^* \mid x_{\mathrm{pa}(d)})} = T_{x,y}(c), \tag{7}$$

where the temperature function $T_{x,y}(c)$ is a function of the updating cycle c and given by[2]

$$T_{x,y}(c) = \frac{1_{\{y \leq c\}}}{x \cdot c}, \quad x \in \mathcal{R}_+,\ y \in \mathcal{N}_+.$$

Figure 4 shows $T_{x,y}$ for different choices of (x, y). From (7), it can be seen that $T_{x,y}$ can be interpreted as the odds of picking a non-optimal move versus an optimal move: In the cth $(c \leq y)$ cycle, it is $c\,x$ times more likely to pick an optimal move than an inferior move. Further, after the yth cycle, the probability of an inferior move drops to zero.

[2]Here, $1_{\{.\}}$ denotes the indicator function. $\mathcal{R}_+$ and $\mathcal{N}_+$ denote the set of positive real numbers and positive integers respectively.

Input:

- A minimal reduction $\mathcal{L}$ with temporal solution order $(d_1, ..., d_n)$;
- Uniform strategy $q_0 := \bar{q}$ and its expected utility $EU(q_0)$;
- $\epsilon > 0$;
- Temperature function $T_{x,y}(c) \in [0, 1]$ decreasing in c;

for $c := 1$ **to** ∞ **do**

1. $q_c := q_{c-1}$;
2. **for** $j := n$ **to** 1 **do**
   (a) **apply** SPU (see Table 1) for decision d_j as follows:
      i. **perform** Step 1–4;
      ii. **perform** Step 5 in which $\tilde{\delta}_{d_j}$ is exchanged by (where α is a normalization constant[a])

$$\tilde{\delta}_{d_j}(x_{d_j} \mid x_{\mathrm{pa}(d_j)}) = \begin{cases} \alpha & \text{if } x_{d_j} = x^*_{d_j} \\ \alpha T_{x,y}(c) & \text{else} \end{cases}$$

      iii. **perform** Step 6;
   (b) **let** $q_c := q_c * \tilde{\delta}_{d_j}$ and **compute** expected utility $EU(q_a)$;
3. **if** $EU(q_a) - EU(q_{a-1}) < \epsilon$ **then stop**.

[a]Here, α equals $[1 + T_{x,y}(c) \cdot (|\mathcal{X}_d| - 1)]^{-1}$ in the case where there is a unique optimal action.

Table 4: The pseudo-code for Simulated Annealing.

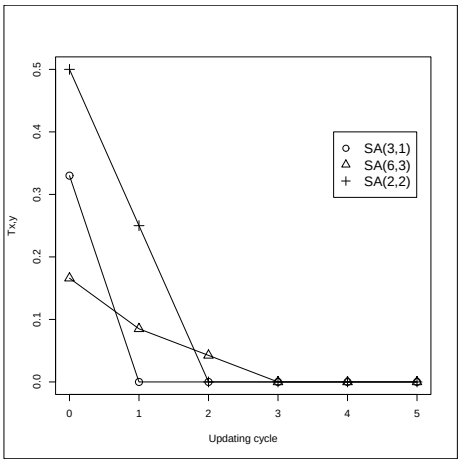

Figure 4: The temperature function $T_{x,y}$ for different parameter choices.

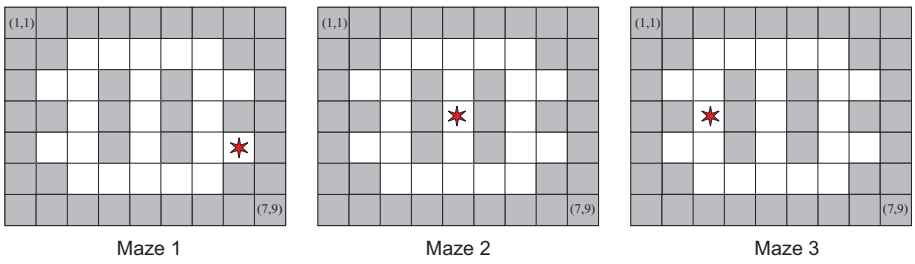

Figure 5: Maze 1–3. Shaded tiles indicate walls and the star denotes the goal state.

Simulated Annealing implemented with $T_{x,y}$ is referred to as SA(x, y). Some comments regarding the implementation of SA(x, y) are needed. Firstly, SA(x, y) always converges to a local maximum strategy since the algorithm eventually behaves like Temporal Policy Updating. Further, it is a deterministic algorithm, i.e. it achieves the same result in two different runs of the algorithm. Secondly, even though each updating step in SA(x, y) is just as computationally efficient as in Temporal Policy Updating, SA(x, y) typically needs a larger number of updates before convergence has occured. Thirdly, during the updating of the policies, it is advantageous to perform partial propagation as explained in Section 3.1. Finally, the algorithm is an iterative improvement algorithm: After each policy updating, the expected utility of the current strategy has increased or is unaltered. This is because the updated policy always performs at least as good as the current policy.

4 Empirical results

This section presents the performance of the various algorithms on the Maze Problem.

4.1 The problems

Figure 5 shows the three mazes in our experiment. The arrangement of walls is the same for the mazes, but the goal state differs. For all experiments, the goal state can always be reached within 10 steps from each starting position.

The algorithms were applied on the LIMID shown in Figure 2. Here, the agent only remembers the current sensor inputs. So, all previous sensor inputs and all previous actions are forgotten and will not be taken into account when an action is taken. Clearly, our LIMID representation of the maze problem is non-soluble because no decision node is extremal (see (2)).

In Maze 1–3, the perfect agent has a global maximum strategy that always finds the goal state without the use of past observations. Table 5 shows such a global maximum strategy for Maze 1.

4.2 The results

The present subsection presents the results of performing the various algorithm on the maze problem, and assess the quality of the obtained local maximum strategies.

Below follow some implementation details of the three algorithms:

- As precision, $\epsilon = 0.0001$ was chosen for all three algorithms.

Decision	Policy
d_1	**if** possible go south; **else if** possible go east; **else** go west;
d_2–d_8	**if** possible go south; **else** go east;
d_9	go north;
d_{10}	go east.

Table 5: Maze 1. A global maximum strategy for the perfect agent. The strategy has 'limited memory' since current sensor inputs are only used.

	MAZE 1				**MAZE 2**				**MAZE 3**				**Mean**
Actuat.	*noi*	*noi*	*per*	*per*	*noi*	*noi*	*per*	*per*	*noi*	*noi*	*per*	*per*	
Sensors	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	
Algorithm													
SA*	.840	.926	.976	1.0	.678	.853	.866	1.0	.800	.958	.977	1.0	.906
Temporal	.838	.899	.964	1.0	.653	.836	.866	1.0	.801	.947	.966	1.0	.897
Greedy	.838	.899	.964	1.0	.625	.620	.645	1.0	.800	.947	.968	1.0	.859
Simulation													
Max	.840	.927	.985	1.0	.684	.859	.934	1.0	.801	.961	.981	1.0	.914
Min	.816	.898	.962	1.0	.275	.323	.464	.455	.682	.872	.810	.909	.705
Mean	.837	.915	.974	1.0	.604	.776	.819	.956	.765	.947	.927	.996	.876

Table 6: Maze 1–3. Expected utilities of strategies obtained by SA*, TPU and Greedy. For each of the 12 experiments, some statistics of 500 random simulated local maximum strategies are also provided.

- In our experiment, Simulated Annealing was implemented using SA(6,3), which henceforth is abbreviated to SA*. As in all applications of simulated annealing, there can be a large number of different annealing schedules. A robustness analysis of the chosen parameters $(x, y) = (6, 3)$ is performed in Section 4.4.

- In TPU and SA* we updated the policies using the (unique) temporal solution order $(d_1, \ldots, d_{10})$ for our LIMID.

To evaluate the quality of the obtained strategies from our three algorithms, we searched for a global maximum strategy for each experiment. This search was done by simulation as follows:[3] Initially, we generated a random solution order for the 10 decision nodes. Then, SPU was applied using the random solution order, to obtain a local maximum strategy. Table 6 displays the results from our experiment:

[3]The simulation study is motivated by the fact that the computation of a global maximum strategy in the non-soluble LIMID representation is computational prohibitive.

	MAZE 1				**MAZE 2**				**MAZE 3**				**Mean**
Actuat.	*noi*	*noi*	*per*	*per*	*noi*	*noi*	*per*	*per*	*noi*	*noi*	*per*	*per*	
Sensors	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	
SA*	0.0	5.8	39.8	0.0	12.0	5.4	41.0	0.0	13.6	37.2	2.6	0.0	13.1
TPU	39.4	95.6	95.2	0.0	37.4	35.4	41.0	0.0	0.0	72.0	11.6	0.0	35.6
Greedy	39.4	95.6	95.2	0.0	52.2	90.2	90.2	0.0	13.6	72.0	7.2	0.0	46.3

Table 7: Maze 1–3. The fraction (in percentage) of the 500 random simulated local maximum strategies whose expected utility exceeds the obtained strategies from SA*, TPU, and Greedy.

Actuators	*noi*	*noi*	*per*	*per*
Sensors	*noi*	*per*	*noi*	*per*
RAF	.7045	.8874	.7767	.8696

Table 8: Maze 1. The results from the Random Access Refinement (RAF) algorithm described in Horsch and Poole (1998).

- In Maze 1, the quality of all simulated local maxima is similar which can be seen from the small difference between the Min and Max values. To some extent, this conclusion also holds for Maze 3.
- In Maze 2, which represents the hardest decision problem, some of the randomly generated local maximum strategies are significantly inferior to the global maximum. For instance, in Maze 2 with the perfect agent, there exists a local maximum strategy that only finds the goal state with probability 45.5%, which is far from the global maximum strategy that always finds the goal state.
- The strategies obtained by SA*, TPU, and Greedy always find the goal state for the perfect agent.

Overall, SA* outperforms the two other algorithms as documented by the following observations from Table 6: Firstly, in 8 out of 9 experiments with non-perfect agents, the SA*-strategy is better or equally good as the best strategy of TPU and Greedy. Further, in our experiments TPU performs better than Greedy. Secondly, in 11 out of the 12 experiments, the SA*-strategy is less than 1% from the simulated maximum. Thirdly, for all 12 experiments, the SA*-strategy performs at least as good as the average local maximum strategy, and typically it performs significantly better.

Table 7 emphasizes the superiority of SA*. The table shows the fraction of the 500 random simulated local maximum strategies whose expected utility exceed the obtained strategies from the three algorithms. On average only 13.1% of the generated local maxima perform better than the SA*-strategy. For comparison, the table shows that on average 35.6% and 46.3% of the generated local maxima outperform the TPU-strategy and Greedy-strategy respectively. Further, in each of the 12 experiments, the SA*-strategy is better than or equal to the majority of the generated local maxima.

Finally, Table 8 displays the results from the anytime algorithm Random Access Refinement (RAF) presented in Horsch and Poole (1998) on the Maze 1 problem. Even though, care should be taken when directly comparing the expected utilities since the numbers for RAF depends on the time that the anytime algorithm was allowed to run, the table gives a strong indication that the SA* algorithm is superiour to the RAF algorithm. In fact all the local maximum strategies computed by simulation perform better than the strategies obtained by RAF.

To conclude, the above analysis shows that Simulated Annealing outperforms the competing algorithms Greedy Search and Temporal Policy Updating and provides significantly better strategies than 'average' local maximum strategies. Section 4.3 investigates the computational aspects of the algorithms. Here, it suffices to note that on average it only took about one second to compute a local maximum strategy using the TPU algorithm, whereas the computational time for SA* and Greedy was slightly longer[4].

[4]The low computational time usage is mainly due to our LIMID representation in which the robot only

	MAZE 1				MAZE 2				MAZE 3				Mean
Actuat.	*noi*	*noi*	*per*	*per*	*noi*	*noi*	*per*	*per*	*noi*	*noi*	*per*	*per*	
Sensors	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	*noi*	*per*	
# updating steps													
SA*	70	50	80	50	80	50	110	50	80	60	90	50	68.3
Temporal	40	20	60	30	70	30	80	20	70	20	60	20	43.3
Greedy	28	10	22	10	21	12	19	10	34	11	28	11	18.0
# flows													
SA*	228	154	235	154	235	154	316	154	235	181	262	154	205.2
Temporal	127	73	181	100	208	100	235	73	208	73	181	73	136.0
Greedy	570	228	456	228	437	266	399	228	684	247	570	247	380.0

Table 9: Maze 1–3. Complexity analysis: The table shows the number of updating steps and the number of flows passed in the junction tree by SA*, TPU, and Greedy. The number of flows approximates the computational complexity of the algorithm.

4.3 Computational aspects

Temporal Policy Updating can be implemented in a simple and efficient way. However, we would claim that Simulated Annealing, and to some extent Greedy Search, can be implemented in a computational feasible manner which is almost as efficient as Temporal Policy Updating. To give further support, this section briefly discuss the computational complexity of the methods when evaluating the various maze problems.

In our analysis, the starting point is the initialized junction tree. The complexity of an algorithm is approximated by the number of flows that are passed in the junction tree until convergence. So, in our complexity analysis we ignore the within root-cliques computations performed when optimizing a policy. These computations are relative unexpensive and will only have little effect on the final result. To see this, note that all root-cliques in the junction tree in Figure 3 are considerably smaller than the other cliques[5].

Table 9 shows the number of updating steps used, and the number of flows passed in the junction tree before convergence. The latter approximates the computational complexity of the algorithm. It can be seen that TPU used the lowest number of flows, whereas SA* passed, on average, only 50% more flows than TPU until convergence. Finally, even though Greedy used fewest updates until convergence, it passed, on average, almost twice as many flows as SA*. This is because, each policy updating in Greedy is done by distributing flows, whereas in TPU and SA* this is done by a simple (partial) collect propagation.

Figure 6 shows the expected utility as a function of updating steps for the three algorithms. As expected, Greedy used the lowest number of updating steps before a local maximum strategy was achieved. After only 21 updating steps Greedy converged, whereas TPU and SA* needed about twice as many updating steps.

4.4 Robustness of Simulated Annealing

As in all applications of simulated annealing, there can be quite a lot of problem-dependent subtlety in the choice of annealing schedule. To investigate the robustness of the output

remembers its most recent observations and actions (see Figure 2).

[5]In the junction tree (Figure 3), the cardinality of the state space of a root-clique is $35 \cdot 5 \cdot 16 = 2800$, whereas for the other cliques the cardinality is $35 \cdot 35 \cdot 5 = 6125$. This is because $|\mathcal{X}_{z_i}| = 7 \cdot 5 = 35$, $|\mathcal{X}_{p_i}| = 2^4 = 16$, and $|\mathcal{X}_{d_i}| = 5$.

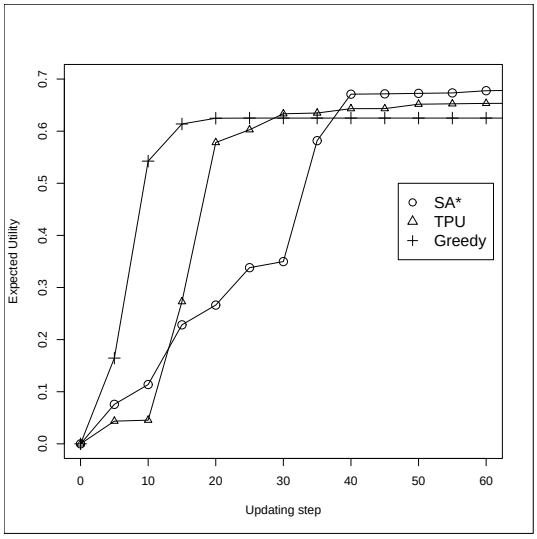

Figure 6: Maze 2(noisy,noisy): Expected utilities as a function of number of updating steps.

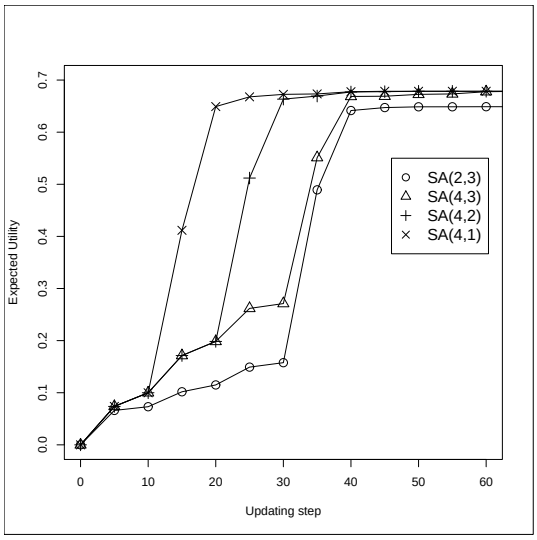

Figure 7: Maze 2(noisy,noisy): Convergence speed of SA(x, y).

strategies, a brute-force sensitivity analysis is performed.

Figure 7 shows the convergence speed of SA(x, y) for different values of (x, y). As ususal, in SA(x, y) it is x times more likely to pick an optimal action than a non-optimal action during the first cycle of updates. Further, non-optimal actions are made during the initial y cycles. As a consequence, the speed of convergence increases as y decreases and x increases.

Table 10-11 present the expected utilities of strategies obtained by 9 different implementations of Simulated Annealing on Maze 2. Below follow some observations: Firstly, all SA-strategies in Table 10 perform well, and significantly better than the average local maxima in Table 11. Secondly, all SA-strategies perform better than the Greedy algorithm. Finally, 19 out of the 21 SA-strategies perform better than the TPU-strategy.

		x						
		2	3	4	5	6	7	8
y	1	0.857	0.857	0.866	0.872	0.873	0.872	0.871
	2	0.867	0.874	0.872	0.874	0.867	0.864	0.865
	3	0.873	0.874	0.872	0.873	0.875	0.866	0.864

Table 10: Maze 1–3. Average utilities obtained by Simulated Annealing SA(x, y) for different values of (x, y). Each number is an average of 9 experiments with non-perfect agents (where each experiment is a combination of a non-perfect agent and a maze).

Algorithm			**Simulation result**		
SA*	TPU	Greedy	Max	Mean	Min
0.875	0.863	0.812	0.886	0.841	0.678

Table 11: Maze 1–3. Average utilities obtained by the algorithms. Each number is an average of 9 experiments with non-perfect agents (where each experiment is a combination of a non-perfect agent and a maze). In addition, statistics of 500 simulations are provided.

5 Conclusion

We presented three methods for evaluating non-soluble LIMIDs. To our knowledge, this is the first paper that performs a careful comparison of methods for evaluating LIMIDs in the non-soluble case. The methods were evaluated on various instances of a large maze problem. To compare the results, for each problem, 500 random local maxima were generated.

Temporal Policy Updating is an instance of SPU proposed in Lauritzen and Nilsson (2001). It performs better than the average local maxima and in addition the obtained strategy using Temporal Policy Updating is easily found. Greedy Search always updates the policy that gives the highest increase in expected utility in the short run. Overall, in our experiments, Greedy Search was on the same level as the average local maxima. However, typically the algorithm performs worse than Temporal Policy Updating, and is almost three times as time-consuming. The final method, Simulated Annealing is inspired by a technique that has attracted significant attention for large scale optimization problems, especially ones where a desired global maximum is hidden among many poorer, local maxima. In our experiments, the algorithm outperforms the above algorithms. It achieved significantly better results than the average local maxima, and in most cases the obtained strategy was close to the maximum of the 500 local maxima. Furthermore, we would claim that Simulated Annealing can be implemented in a computational feasible manner: In the experiments of the various maze problems, our implementation of Simulated Annealing was less than twice as time-consuming as Temporal Policy Updating.

Future research may lead to additional refinements of the algorithms proposed in this paper. For instance, as in all applications of simulated annealing there can be quite a lot of problem-dependent subtlety in the choice of annealing schedule and we anticipate that refinements will be investigated.

References

Bangsø, O. and Wuillemin, P.-H. (2000). Top-down Construction and Repetitive Structures Representation in Bayesian Networks. pp. 282–6. AAAI Press.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., and Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York.

Höhle, M., Jørgensen, E., and Nilsson, D. (2000). Modeling with LIMIDs - Examplified by Disease Treatment in Slaughter Pigs. In *Proceedings of the International Symposium on Pig Herd Management Modeling and Information Technologies Related*, (ed. L. Plà and J. Pomar), pp. 17–26.

Horsch, M. C. and Poole, D. (1998). An anytime algorithm for decision making under uncertainty. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, (ed. G. F. Cooper and S. Moral), pp. 246–55. Morgan Kaufmann, San Francisco.

Jensen, F., Jensen, F. V., and Dittmer, S. L. (1994). From influence diagrams to junction trees. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, (ed. R. L. de Mantaras and D. Poole), pp. 367–73. Morgan Kaufmann Publishers, San Francisco, CA.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer-Verlag, New York.

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

Koller, D. and Pfeffer, A. (1997). Object-oriented Bayesian networks. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*, (ed. D. Geiger and P. P. Shenoy), pp. 302–13. Morgan Kaufmann Publishers, San Francisco.

Lauritzen, S. L. and Nilsson, D. (2001). LIMIDs of Decision Problems. *Management Science*, **47**, 1238–51.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, (ed. A. Prieditis and S. Russell), pp. 362–70. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, San Francisco, CA.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, **28**, (1–4), 47–66.

Madsen, A. and Nilsson, D. (2001). Solving Influence Diagrams using HUGIN, shafer-shenoy, and lazy propagation. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Inteligence*, (ed. J. Breese and D. Koller), pp. 337–45. Morgan Kaufmann Publishers, San Francisco, California.

Nielsen, T. D. and Jensen, F. V. (1999). Welldefined decision scenarios. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pp. 502–11. Morgan Kaufmann Publishers, San Francisco, CA.

Nilsson, D. and Lauritzen, S. (2000). Evaluating influence diagrams using LIMIDs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI-2000)*, pp. 436–45. Morgan Kaufmann Publishers, San Francisco, CA.

Pearl, J. (1986). A constraint–propagation approach to probabilistic reasoning. In *Uncertainty in Artificial Intelligence*, (ed. L. N. Kanal and J. F. Lemmer), pp. 357–70. North-Holland, Amsterdam, The Netherlands.

Shachter, R. (1998). Bayes-ball: The rational pasttime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pp. 48–487. Morgan Kaufmann Publishers, San Francisco, CA.

Shafer, G. R. and Shenoy, P. P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence*, **2**, 327–52.

Shenoy, P. P. (1992). Valuation-based systems for Bayesian decision analysis. *Operations Research*, **40**, 463–84.

Shenoy, P. P. and Shafer, G. R. (1990). Axioms for probability and belief–function propagation. In *Uncertainty in Artificial Intelligence 4*, (ed. R. D. Shachter, T. S. Levitt, L. N. Kanal, and J. F. Lemmer), pp. 169–98. North-Holland, Amsterdam, The Netherlands.

Verma, T. and Pearl, J. (1990). Causal networks: Semantics and expressiveness. In *Uncertainty in Artificial Intelligence IV*, (ed. R. D. Schachter, T. S. Levitt, L. N. Kanal, and J. F. Lemmer), pp. 69–76. North-Holland, Amsterdam.