International Journal of Approximate Reasoning 68 (2016) 230–245

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

# Fast local search methods for solving limited memory influence diagrams

Denis Deratani Mauá [a,*], Fabio Gagliardi Cozman [b]

[a] Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, São Paulo 05508-090, Brazil
[b] Escola Politécnica, Universidade de São Paulo, Av. Prof. Mello Moraes, 2231, São Paulo 05508-030, Brazil

ARTICLE INFO

*Article history:*
Received 3 January 2015
Received in revised form 5 May 2015
Accepted 6 May 2015
Available online 13 May 2015

*Keywords:*
Influence diagrams
Probabilistic graphical models
Decision making
Local search
Parameterized complexity

ABSTRACT

Limited memory influence diagrams are graph-based models that describe decision problems with limited information such as planning with teams and/or agents with imperfect recall. Solving a (limited memory) influence diagram is an NP-hard problem, often approached through local search. In this work we give a closer look at $k$-neighborhood local search approaches. We show that finding a $k$-neighboring strategy that improves on the current solution is W[1]-hard and hence unlikely to be polynomial-time tractable. We also show that finding a strategy that resembles an optimal strategy (but may have low expected utility) is NP-hard. We then develop fast schema to perform approximate $k$-local search; experiments show that our methods improve on current local search algorithms both with respect to time and to accuracy.



## 1. Introduction

Limited memory influence diagrams (LIMIDs) are graph-based probabilistic decision making models tailored for situations involving teams and limited-resource agents [10,22,34]. LIMIDs relax the *perfect recall* requirement (a.k.a. no-forgetting assumption) of traditional influence diagrams [18], and by doing so, require considerably more computational effort in the search for optimal policies.

Finding an optimal strategy for polytree-shaped LIMIDs is NP-hard even if variables are ternary and the utility function is univariate [26], or if variables are binary and the utility function is multivariate [27]. Similar negative results hold for approximating the problem within any fixed constant when either variable cardinality or treewidth is unbounded [26]. And even though there are polynomial-time approximations when cardinalities and treewidth are bounded [25], constants in such solutions are so big as to prevent practical use. Currently the state-of-art algorithm for solving LIMIDs exactly is Multiple Policy Updating (MPU) [24], that works by verifying a dominance criterion between partial strategies. MPU has worst-case exponential cost but often finishes in reasonable time. There are also anytime solvers based on branch-and-bound that can trade off accuracy for efficiency [8,9,19].

In practice, local search methods are the most widely used algorithms for approximately solving LIMIDs. Lauritzen and Nilsson [22] developed the Single Policy Updating (SPU) algorithm for computing locally optimum strategies in arbitrary LIMIDs. Their algorithm remains the most referenced algorithm for solving medium and large LIMIDs. SPU iteratively seeks for a variable that can improve the incumbent global strategy by modifying its associated actions. If no such variable is

* Corresponding author.
*E-mail addresses:* denis.maua@usp.br (D.D. Mauá), fgcozman@usp.br (F.G. Cozman).

http://dx.doi.org/10.1016/j.ijar.2015.05.003
0888-613X/

found the algorithm halts at a local optimum. Thus, SPU essentially implements a 1-neighbor local search in the space of strategies. Detwarasiti and Shachter [10] extended SPU to allow larger moves in the search space. Roughly speaking, their approach can be seen as a $k$-neighbor local search in the space of strategies. In practice, exhaustive $k$-local search can only be applied to networks with say less than $n = 100$ decision variables and with very small values of $k$ (say, $k = 2$), as each step requires exploration of $O(c^k n^k)$ candidates, where $c$ is an upper bound on the cardinality of the action variables. Watthayu [33] and Liu and Ihler [23] proposed message-passing algorithms to cope with high treewidth diagrams. Their algorithms can also be seen as performing a 1-local search in the space of strategies (especially when action nodes have no parents).

In this paper we focus on the local search that runs within the methods in the last paragraph. Borrowing arguments from parameterized complexity theory, we prove that $k$-local search cannot be realized within time $O(f(k)n^\alpha)$ unless FPT equals W[1]-hard (a strongly disbelieved equality), where $f$ is an arbitrary computable function and $\alpha$ is a constant independent of $k$ (Section 3). We take such a result as an indication that exploring entire $k$-neighborhoods is infeasible for large values of $k$ and $n$, and that relaxed methods such as stochastic search ought to be used. But even if we randomly sample a subset of action variables of cardinality $k$, a naive approach still has to consider $O(c^k)$ candidates.

We thus investigate the use of MPUs pruning to speed up SPU and related $k$-local search schema (Section 4). We propose a relaxed and approximate version of MPUs pruning, with worst-case polynomial-time complexity (Section 5). This approximate pruning method is used in each $k$-local search, leading to very efficient versions of local search methods for LIMIDs. We prove that when $k$ is the number of action variables, our approximate pruning provides an additive fully polynomial-time approximation scheme for LIMIDs of bounded treewidth and bounded variable cardinality. Finally, we show by experiments with random and realistic diagrams that our local search algorithms, both exact and approximate, outperform existing local search methods (Section 6).

## 2. Limited memory influence diagrams

LIMIDs are graphical representations of structured decision problems [17]. Variables in a decision problem can be partitioned into state (or chance) variables $\mathcal{S}$, which represent quantities unknown at planning stage, action (or decision) variables $\mathcal{A}$, which enumerate alternative courses of action, and value variables $\mathcal{V}$, which associates actions and states with rewards. We assume here that variables take on finitely many values. Each variable in a decision problem represented as a LIMID is equated with a node in a directed acyclic graph; in particular, value variables are equated to leaf nodes. There is a (conditional) probability distribution $P(S|\mathcal{P}_S)$ for every state variable $S \in \mathcal{S}$, where the notation $\mathcal{P}_X$ denotes the parents of a variable $X$ in the graph. There is also a utility function $U(\mathcal{P}_V)$ for every value variable. The overall utility $U$ is assumed to decompose additively in terms of the value variables [32], that is, $U(\mathcal{S}, \mathcal{A}) = \sum_{V \in \mathcal{V}} U(\mathcal{P}_V)$. State variables are assumed to satisfy the Markov condition, which states that any (state) variable is independent of its non-descendant non-parents conditional on its parents. Consequently, the joint distribution of state variables conditioned on a configuration $\mathcal{A} = a$ of the action variables factorizes as $P(\mathcal{S}|\mathcal{A} = a) = \prod_{S \in \mathcal{S}} P(S|\mathcal{P}_S, \mathcal{A} = a)$.

A strategy $\delta = \{\delta_A : A \in \mathcal{A}\}$ is a multiset of local decision rules, or policies, one for each action variable in the problem. Each policy $\delta_A$ is a mapping from the configurations of the values of the parents $\mathcal{P}_A$ of $A$ to values of $A$. We denote by $\Delta_A$ the set of all policies for variable $A$. A policy for an action variable with no parents is simply an assignment of a value to that variable. We assume that policies are encoded as tables. Hence, the size of a policy is exponential in the number of parents of the corresponding action variable, which in real scenarios forces us to constrain the maximum number of parents of an action node lest the implementation of a policy be impractical.

Given an action variable $A$ and a policy $\delta_A$, we let $P(A|\mathcal{P}_A, \delta_A)$ be the collection of degenerate conditional probability distributions that assign all mass to $a = \delta_A(\mathcal{P}_A)$ (or the degenerate marginal distribution $P(A|\delta_A)$ that places all mass on $\delta_A$ in case $A$ has no parents). With this correspondence between policies and (conditional) probability distributions, we can define a joint probability distribution over the state and action variables for any given strategy $\delta$ as

$$P(\mathcal{S}, \mathcal{A}|\delta) = \prod_{S \in \mathcal{S}} P(S|\mathcal{P}_S) \prod_{A \in \mathcal{A}} P(A|\mathcal{P}_A, \delta).$$

The expected utility of a strategy $\delta$, $E(U|\delta)$, is then $\sum_{\mathcal{S},\mathcal{A}} U(\mathcal{S}, \mathcal{A}) P(\mathcal{S}, \mathcal{A}|\delta)$.

Given a strategy $\delta$, computing $E(U|\delta)$ can be reduced in polynomial-time to a marginal inference in a Bayesian network [6]. Conversely, marginal inference in Bayesian networks, a $\#P$-complete problem [30], can be reduced in polynomial time to the computation of an expected utility by using a $\{0, 1\}$-valued utility and making the conditional probabilities of the children of action nodes numerically independent of strategies. Hence, those two problems are computationally equivalent. Marginal inference can be performed in time exponential in the treewidth of the underlying graph by e.g. variable elimination. This entails a polynomial-time algorithm when treewidth is small and considered constant in the complexity analysis. Kwisthout et al. [21] showed that under the widely believed hypothesis that SAT is not subexponential-time solvable, variable elimination's performance is optimal. Thus it seems necessary to constrain LIMIDs to bounded treewidth diagrams if worst-case efficient computations are sought (it is possible that the average cost of marginal inference is polynomial; we do not pursue this possibility here).

### 2.1. Decision making with LIMIDs

An important task with LIMIDs is that of deciding whether the maximum expected utility exceeds a given threshold:

**Decide Maximum Expected Utility (DMEU)**
*Input:* A LIMID and a rational $k$
*Question:* Is there a strategy $\delta$ such that $E(U|\delta) \geq k$?

DMEU is $\mathrm{NP}^{\mathrm{PP}}$-complete, and NP-complete when restricted to diagrams of bounded treewidth [8]. The problem is NP-complete on LIMIDs of bounded treewidth even when all variables are binary [27], and when all variables are ternary and there is a single value node [26].

We can compute the value of the maximum expected utility by binary search in polynomial time if DMEU can be solved in polynomial time. Similarly, if the in-degrees of action nodes are bounded we can use a polynomial-time algorithm $\mathcal{M}$ that solves DMEU to obtain an optimal strategy in polynomial time as follows. First, perform a binary search using $\mathcal{M}$ to compute the maximum expected utility and use that value as $k$. Select an action variable $A$ and for every policy $\delta_A$ build a new LIMID where $A$ is a state node with conditional probability $P(A|\delta_A)$. Now run the algorithm $\mathcal{M}$ on those LIMIDs: the algorithm will certainly return a yes answer on some of them; any policy $\delta_A$ corresponding to an affirmative answer is part of the optimal strategy, and we can repeat the procedure for another action variable until no action variables remains. Conversely, assuming the same requirements on the representation of expected utilities and strategies, DMEU can be efficiently solved by any polynomial-time algorithm that computes the maximum expected utility. Finally, if the treewidth of the diagrams is bounded, DMEU can trivially be solved in polynomial-time by any polynomial-time algorithm that finds optimal strategies. Thus, DMEU is largely equivalent to selecting an optimal strategy and computing the value of the maximum expected utility.

We make extensive use of the following result that follows immediately from the results of Mauá, de Campos and Zaffalon [25,26].

**Theorem 1.** *Given a LIMID $\mathcal{L}$ of treewidth $w$ we can construct in time polynomial in its size a LIMID $\mathcal{L}'$ and a bijective function $f$ such that*

(i) *$\mathcal{L}'$ has a single value variable $V$ such that $0 \leq U(\mathcal{P}_V) \leq 1$, and treewidth at most $w+3$;*
(ii) *the action nodes in $\mathcal{L}'$ have no parents;*
(iii) *$f$ maps strategies $\delta'$ of $\mathcal{L}'$ into strategies $\delta$ of $\mathcal{L}$ in linear time;*
(iv) *if $\delta'$ is a strategy for $\mathcal{L}'$ and $\delta = f(\delta')$ then $E(U|\delta) = k_1 E(U'|\delta') + k_2$, where $U$ and $U'$ denote the utility functions of $\mathcal{L}$ and $\mathcal{L}'$, respectively, and $k_1$ and $k_2$ are constants computed in linear time.*

A corollary of the above result is that the maximum expected utility of $\mathcal{L}'$ equals the maximum expected utility of $\mathcal{L}$ up to a constant, and the optimal strategy for $\mathcal{L}$ can be obtained from the optimal strategy for $\mathcal{L}'$. Hence, unless when explicitly mentioned, we assume in the rest of the paper that LIMIDs have a single value node taking its values in $[0,1]$, and that action nodes are parentless.

Fig. 1 shows the graphical structure of a LIMID with non-root action nodes and multiple value nodes (on the left), and its corresponding LIMID with parentless action nodes and single value variable (on the right) obtained by application of the transformations described in [25,26]. The action variables are assumed binary. The non-root action variable $C$ is transformed into a chance variable on the right. The dependence of the action variable $C$ in the LIMID on the left on $B$ and $D$ is obtained in the LIMID on the right with the subnetwork containing the action variables $C_1, \ldots, C_4$ and the chance variables $I_1, \ldots, I_4$; these variables take values in the domain of $C$. The conditional distribution $P(C|I_4)$ equals one if $C = I_4$ and zero otherwise. Consider an arbitrary ordering $\pi_1, \ldots, \pi_4$ of the joint values of $B$ and $D$. We define

$$P(I_1|C_1, B, D) = \begin{cases} 1, & \text{if } (B,D) = \pi_1 \text{ and } I_1 = C_1\,, \\ 1/4, & \text{if } (B,D) \neq \pi_1, \\ 0, & \text{otherwise,} \end{cases}$$

and, for $i = 2, 3, 4$,

$$P(I_i|I_{i-1}, C_i, B, D) = \begin{cases} 1, & \text{if either } (B,D) \neq \pi_i \text{ and } I_i = I_{i-1} \\ & \text{or if } (B,D) = \pi_i \text{ and } I_i = C_i\,, \\ 0, & \text{otherwise.} \end{cases}$$

The effect of these specifications is that the induced distribution $P(C|C_1, \ldots, C_4, B, D)$ acts as a multiplexer that for input $(B,D) = \pi_i$ and $C_1, \ldots, C_4$ assigns all mass on $C = C_i$, hence replicating the decision rule $\delta_C$. The multiple value nodes in the LIMID on the left are transformed into binary chance variables and connected to the binary variable $O$ (when the number of value variables is high, they can be connected as a chain or binary tree in order to keep the in-degree small); their conditional probabilities are

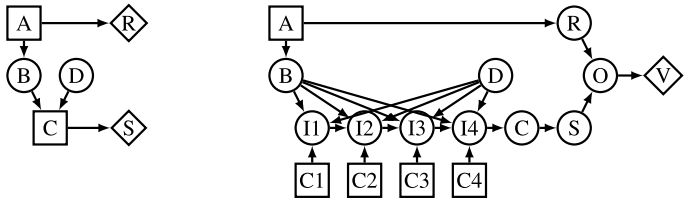


**Fig. 1.** A LIMID representing sequential decisions and multivariate utility function (on the left) and a strategy-equivalent LIMID with parentless action nodes and univariate utility (on the right) obtained by application of Theorem 1.

$$P(S=1|C)=\frac{U(C)-u_0}{u_1-u_0}\,,\qquad P(R=1|A)=\frac{U(A)-u_0}{u_1-u_0}\,,$$

and

$$P(O=1|S,R)=\begin{cases}1, & \text{if } S=R=1\,,\\ 0, & \text{if } S=R=0\,,\\ 1/2, & \text{if } S\neq R\,,\end{cases}$$

where $u_0=\min\{\min_c U(C=c),\min_a U(A=a)\}$ and $u_1=\max\{\max_c U(C=c),\max_a U(A=a)\}$. The value variable $V$ on the right is associated with the utility function

$$U'(O=1)=1\,,\qquad U'(O=0)=0\,.$$

The function $f$ maps a strategy $\delta'=\{\delta_A,\delta_{C_1},\ldots,\delta_{C_4}\}$ for the right-hand side diagram into a strategy $f(\delta')=\delta=\{\delta_A,\delta_C\}$ such that $\delta'_{C_i}=\delta_C(\pi_i)$. One can check that

$$E(U(C)+U(A)|f(\delta'))=2(u_1-u_0)E(U'(O)|f(\delta))+u_0$$

for any strategy $\delta'$, which agrees with Theorem 1.

## 3. The complexity of *k*-neighborhood local search

Consider a strategy $\delta$ and a set of action variables $\mathcal{N}\subseteq\mathcal{A}$. The $\mathcal{N}$-neighborhood of $\delta$ is the set of strategies $\delta'$ that coincide with $\delta$ on the policies of variables $A\in\mathcal{A}\setminus\mathcal{N}$. A $k$-neighbor of $\delta$ is any strategy in a $\mathcal{N}$-neighborhood of $\delta$ with $|\mathcal{N}|=k$. The $k$-neighborhood of $\delta$ is the set of its $k$-neighbors. Arguably, the most widely used scheme for selecting strategies is $k$-Policy Updating, which operates as follows.

**Algorithm 1** $k$-Policy Updating ($k$PU).

**Require:** A LIMID, a strategy $\delta_0$ and a positive integer $K$.
  **for** $i$ taking values 1 **to** $K$ **do**
    **if** there exists a $k$-neighbor $\delta_i$ of $\delta_{i-1}$ such that $E(U|\delta_i)>E(U|\delta_{i-1})$ **then**
      select such a strategy
    **else**
      **return** $\delta_{i-1}$
    **end if**
  **end for**
  **return** $\delta_M$.

For large enough and finite $K$ the procedure converges to a local optimum. In particular, if $k$ equals the number of action variables the algorithm finds a global optimum. In terms of speed, the main bottleneck of $k$PU is the $k$-local search step, where an improving solution is searched for; this can be formalized as

***k*-Policy Improvement (*k*PI)**
*Input:* A LIMID $\mathcal{L}$ and a strategy $\delta$
*Parameter:* A positive integer $k$
*Question:* Is there a $k$-neighbor $\delta'$ of $\delta$ such that $E(U|\delta')>E(U|\delta)$?

The same argument used when discussing DMEU can be used here to show that the problem of finding the maximum expected utility in the $k$-neighborhood of a strategy and the problem of selecting a $k$-neighbor with higher expected utility (if it exists) are largely equivalent to $k$PI in the sense that (under the same requirements) a polynomial-time algorithm for one problem can be used to solve the other.

For a LIMID whose action variables are parentless, $k$PI can be solved by exhaustive search in the $k$-neighborhood in time $O(n^k c^k)$, where $n=|\mathcal{A}|$ is the number of action variables and $c$ is the maximum cardinality of an action variable. Such an

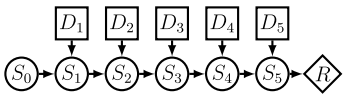


**Fig. 2.** Chain-like LIMID used to prove Theorem 2.

approach is prohibitive for large values of $n$ or $c$ and moderate values of $k$. It is thus interesting to look for faster methods for searching the $k$-neighborhood of a strategy. In particular, we should ask whether there is an algorithm that runs in time $O(f(k)b^{\alpha})$, where $f$ is an arbitrary computable function, $b$ is the size of the LIMID (encoded as a bitstring), and $\alpha$ is a constant independent of $k$. In other words, we are interested in knowing whether it is possible to scale up $k$-local search to diagrams with hundreds of variables if $k$ is kept small. We now show that finding such an algorithm implies that FPT = W[1], and is therefore unlikely. To this aim, we need to introduce some background on the rich field of parameterized complexity [11–13].

Parameterized complexity investigates the runtime behavior of inputs $(x, k)$ that can be decomposed into two parts, its main part $x$ and a parameter $k$. Many interesting NP-hard problems are polynomial-time solvable for fixed values of the parameters, that is, when the parameter is not taken to be part of the input. There are essentially two kinds of polynomial runtime for fixed parameters. A (decision) problem is said to be *fixed-parameter tractable* if there is an algorithm that solves any parameterized instance $(x, k)$ of the problem in time $O(f(k)b^{\alpha})$, where $f$ is an arbitrary computable function, $b$ is the size of the input and $\alpha$ is a constant that does not depend on $k$ [11]. The class of all fixed-parameter tractable decision problems is denoted FPT. Note that NP-complete problems can be either fixed-parameter tractable or intractable.

Similar to the polynomial hierarchy in the NP-completeness framework, the family W[$t$] defines a hierarchy of nested and increasingly more complex parameterized problems for $t = 1, 2, \ldots$. Roughly speaking, W[$t$] is the class of parameterized problems that can compute Boolean circuits of depth at most $t$. We have that FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots$. It is unknown whether any of these inclusions is proper, but there are good reasons to believe that at least the first inclusion (i.e., FPT $\subseteq$ W[1]) is proper [12]: if FPT = W[1] then NP-complete problems can be solved in subexponential time [13].

Instead of polynomial-time reductions, which one uses to show NP-hardness of non-parameterized problems, fixed-parameter intractability is usually shown by many-one parameterized reductions to W[$t$]-hard problems. A many-one parameterized reduction from a problem $A$ to problem $B$ takes an instance $(x, k)$ of $A$ and produces an instance $(x', g(k))$ of $B$ in time $O(f(k)b^{\alpha})$, where $g$ and $f$ are arbitrary computable functions, $b$ is the length of $x$ and $\alpha$ is a constant.

The next result shows that if a fixed-parameter tractable algorithm that performs $k$-local search in the space of strategies existed we would prove $k$-FLIP MAX SAT to be fixed-parameter tractable, implying that W[1] = FPT.

**Theorem 2.** *Unless W[1] = FPT, there is no algorithm that solves k-PI in time $O(f(k)b^{\alpha})$, where b is the size of the bitstring encoding of the LIMID, $f$ is an arbitrary computable function and $\alpha$ is a constant independent of k, even for polytree-shaped LIMIDs of bounded treewidth.*

**Proof.** We use a parameterized reduction from $k$-FLIP MAX SAT to prove the result; that is, we consider the following variant of MAX SAT:

**$k$-Flip Max Sat**
*Input:* A CNF formula $F$ and a truth-value assignment $\tau_0$
*Parameter:* A positive integer $k$
*Question:* Is there a $k$-flip of $\tau_0$ satisfying more clauses of $F$ than $\tau_0$?

A $k$-flip of an assignment $\tau$ is another assignment that differs from $\tau$ in the values assigned to at most $k$ variables. Szeider [31] showed that the above problem is W[1]-hard.

Consider formula $F$, truth assignment $\tau$ and parameter $k$, and let $X_1, \ldots, X_n$ be the variables in $F$, and $C_1, \ldots, C_m$ be its clauses. We build a corresponding LIMID with graph structure as in Fig. 2 and numerical parameters specified as follows (this construction is similar to the construction used by [29] to show NP-hardness of MAP inference in polytree-shaped Bayesian networks). The variables $S_1, \ldots, S_n$ take values in $\{0, 1, \ldots, m\}$. The variable $S_0$ takes values in $\{1, \ldots, m\}$, and the action variables are binary and take on values 0 and 1. The conditional probabilities of the chance variables are specified as follows:

$$P(S_i|S_{i-1}, D_i) = \begin{cases} 1 & \text{if } S_i = S_{i-1} = 0, \\ 1 & \text{if } S_i = 0 \text{ and } S_{i-1} = k \geq 1 \text{ and } D_i \text{ satisfies } C_k, \\ 1 & \text{if } S_i = S_{i-1} = k \geq 1 \text{ and } D_i \text{ does not satisfy } C_k, \\ 0 & \text{otherwise,} \end{cases}$$

and $P(S_0 = s_0) = 1/m$ for all $s_0$. The utility is defined as $U(S_n = 0) = 1$ and $U(S_n = s_n) = 0$ for all $s_n \neq 0$. The variable $S_0$ serves as a clause selector: $S_0 = i$ denotes that clause $i$ is being selected. Variable $S_i$, $i = 1, \ldots, n$, indicates whether the clause selected by $S_0$ is satisfied by some of $D_1, \ldots, D_i$. We then have that $E(U|\delta) = \#\mathsf{SAT}(\delta)/m$, where #SAT is the number of clauses satisfied by a truth-value assignment corresponding to $\delta$. Consider an arbitrary strategy $\delta$ corresponding to the

truth-value assignment $\tau$ (i.e., $\delta_{D_i} = 1$ if $\tau(X_i)$ is true, and $\delta_{D_i} = 0$ otherwise). A $k$-neighbor of $\delta$ is a strategy $\delta'$ differing from $\delta$ in at most $k$ coordinates. Hence, there is a truth-value assignment satisfying more clauses in $F$ than $\tau$ if and only there is $k$-policy improvement of $\delta$, and the result is proved. □

Another factor that affects the performance of $k$PU is the initial strategy $\delta_0$. If we could efficiently find an initial strategy that is $k$-neighbor of an optimum strategy then we could find an optimal strategy in time $O(c^k)$, where $c$ is an upper bound in the cardinality of action variables. Note that a $k$-neighbor of an optimal strategy can have low expected utility, hence finding a $k$-neighbor is not the same as finding a high expected utility strategy. However, we can show that:

**Corollary 1.** *Finding a k-neighbor of an optimum strategy for any k strictly smaller than half of the number of action variables is NP-hard.*

**Proof.** In [15] the authors showed that finding a truth-value assignment to 3SAT that is a $k$-neighbor of a satisfying assignment is NP-hard. Consider the reduction in the proof of Theorem 2. There is a one-to-one mapping between strategies of the LIMID and assignments of the SAT problem instance. Moreover, an assignment of the SAT instance is satisfying if and only if the corresponding strategy has expected utility equal to one (i.e., if all clauses are satisfied). Since this is the maximum value a strategy can achieve (because $0 \leq U \leq 1$), satisfying assignments map into optimum strategies and vice-versa. Hence, if we can efficiently find a $k$-neighbor of an optimal strategy then we can efficiently find an assignment that is a $k$-neighbor of a satisfying one. □

The result above shows that not only finding high quality strategies is a difficult problem, but that finding strategies that "resemble" high quality strategies is also hard. Moreover, it suggests that randomly choosing an initial strategy is the best we can hope on large instances (if we aim at efficiency).

## 4. Improving $k$-policy updating: D$k$PU

The result in the previous section indicates that local search becomes difficult once we try to refine search by increasing its width (through $k$). Thus we must focus on approximate ways that allow us to climb up to reasonably large $k$ (say, 10) for large values of $n$ (where $n$ is the number of action variables).

Assuming without loss of generality that a LIMID has parentless action variables of cardinality $c$, a brute-force approach to $k$-local search can be accomplished by examining the $\binom{n}{k}$ subsets $\mathcal{N} \subseteq \mathcal{A}$ of cardinality $k$, and for each $\mathcal{N}$ examining all the $c^k$ joint configurations of variables $\mathcal{N}$. Hence, there are two sources of inefficiency in this approach: finding $\mathcal{N}$ and selecting an $\mathcal{N}$-neighbor. We tackle the first problem by randomly sampling sets $\mathcal{N}$, which guarantees uniform coverage. The search for $\mathcal{N}$-neighbors that improve on the current best strategy found is more intricate. In this section we develop a fast procedure for selecting the *optimal* $\mathcal{N}$-neighbor of an incumbent strategy for a fixed $\mathcal{N}$ (an optimal neighbor is one that maximizes the expected utility among all neighbors).

### 4.1. Dominance pruning

We start with some basic concepts. First, the notion of a potential:

**Definition 1.** A *potential* $\phi$ with *scope* $\mathcal{X}$ is a nonnegative real-valued mapping of configurations $x$ of the set of variables $\mathcal{X}$.

We assume the usual algebra of potentials: the product $\phi \cdot \psi$ of potentials $\phi$ and $\psi$ returns a potential $\gamma$ on $z \sim \mathcal{Z} = \mathcal{X} \cup \mathcal{Y}$ such that $\gamma(z) = \phi(x) \cdot \psi(y)$, where $x$ and $y$ are the projections of $z$ onto $\mathcal{X}$ and $\mathcal{Y}$, respectively. The $\mathcal{Y}$-marginal $\sum_{\mathcal{X}\setminus\mathcal{Y}} \phi$ of a potential $\phi$ with scope $\mathcal{X}$, where $\mathcal{Y} \subseteq \mathcal{X}$, is the potential $\psi$ with scope $\mathcal{Y}$ such that $\psi(y) = \sum\{\phi(x) : x \sim y\}$.

The system of potentials with product and marginalization forms a valuation algebra [20]. This entails that given a set of potentials $\Gamma$ a marginal $\sum_{\mathcal{X}} \prod_{\psi\in\Gamma} \psi$ can be computed by variable elimination: for each variable $X \in \mathcal{X}$ in some ordering, remove from $\Gamma$ all potentials whose scope contain $X$, compute the marginal of those products which sums over $X$ and add the result to $\Gamma$.

Since by definition the probability and utility functions in a LIMID are potentials, the expected utility of a given strategy can be computed by variable elimination. The potentials produced during variable elimination satisfy the following property, which we use later on:

**Proposition 1.** *Consider a LIMID with a single value variable $V$ and a strategy $\delta$, and let $\Gamma = \{P(S|\mathcal{P}_S), P(A|\delta_A), U(\mathcal{P}_V) : S \in \mathcal{S}, A \in \mathcal{A}\}$. If $0 \leq U(\mathcal{P}_V) \leq 1$, then every potential $\phi$ generated during variable elimination satisfies $0 \leq \phi \leq 1$.*

**Proof.** Assume an elimination order $<$ (smaller variables are eliminated first), and consider the step in variable elimination where all potentials containing a variable $X$ are collected. Denote by $\mathcal{Y}$ the set of variables $Y < X$ (i.e., the set of variable that

have been previously eliminated), and for any variable $Y$ let $F_Y$ denote $Y$ and its children. By design, the only potentials in the initial $\Gamma$ whose scope include a variable $Y$ are $P(Y|\mathcal{P}_Y)$ (or $U(\mathcal{P}_Y)$ if $Y = V$) and $P(Z|\mathcal{P}_Z)$ for $Z \in F_Y$. By the properties of a valuation algebra, it follows that $\psi_X = \sum_X P(X|\mathcal{P}_X) \sum_{\mathcal{Y}} \prod_{Z \in F_Y : Z \neq X, Y \in \mathcal{Y}} P(Z|\mathcal{P}_Z)$, where $P(Z|\mathcal{P}_Z)$ is $U(\mathcal{P}_Z)$ if $Z = V$. The right-hand side of the equality is a convex combination of $\sum_{\mathcal{Y}} \prod_{Z \in F_Y : Z \neq X, Y \in \mathcal{Y}} P(Z|\mathcal{P}_Z)$, and therefore is a function not greater or smaller than that in every coordinate. The result follows by induction. □

The algebra of potentials can be extended to set-valued objects, so as to obtain maximum expected utility and hence solve DMEU [24]. To do so, we define:

**Definition 2.** A *set-potential* $\Phi(\mathcal{X})$ is a set of potentials $\phi$ with scope $\mathcal{X}$.

**Definition 3.** The product of two set-potentials $\Phi(\mathcal{X})$ and $\Psi(\mathcal{Y})$ is the set-potential

$$[\Phi \cdot \Psi](\mathcal{X} \cup \mathcal{Y}) = \{\phi \cdot \psi : \phi \in \Phi, \psi \in \Psi\}.$$

**Definition 4.** The $\mathcal{Y}$-marginal of a set-potential $\Phi(\mathcal{X})$ with respect to a variable set $\mathcal{Y} \subseteq \mathcal{X}$ is the set-potential

$$\left[\sum_{\mathcal{X}\setminus\mathcal{Y}} \Phi\right](\mathcal{Y}) = \left\{\sum_{\mathcal{X}\setminus\mathcal{Y}} \phi : \phi \in \Phi\right\}.$$

Mauá, de Campos and Zaffalon [26] proved that the algebra of set-potentials is a valuation algebra [20], and thus marginal inference with set-potentials can also be computed by variable elimination (with potentials and their operations replaced by their set counterparts).

The product of set-potentials creates exponentially large set-potentials. Our interest in the algebra of set-potentials is, as we show later on, to select one single table in a set-potential produced by marginalization and product of many set-potentials. As most of the tables produced are irrelevant, we can save computations by pruning them from set-potentials generated during variable elimination. One way of doing this is by defining a dominance criterion between potentials:

**Definition 5.** Consider two potentials $\phi(\mathcal{X})$ and $\psi(\mathcal{X})$ with the same scope. We say that $\phi$ dominates (resp., is dominated by) $\psi$ if $\phi(x) \geq \psi(x)$ (resp., $\phi(x) \leq \psi(x)$) for all $x$.

We can define the set of non-dominated potentials:

**Definition 6.** The dominance-pruning of a set-potential $\Phi(\mathcal{X})$ is the set-potential $\mathsf{nd}[\Phi]$ of non-dominated potentials in $\Phi$.

Note that if $\mathsf{nd}$ is applied on a set-potential with empty scope, it produces a single real number. Mauá, de Campos and Zaffalon [26] showed that dominance-pruning satisfies

$$\mathsf{nd}[\Phi(\mathcal{X})\Psi(\mathcal{Y})] = \mathsf{nd}\Big[\mathsf{nd}[\Phi(\mathcal{X})]\mathsf{nd}[\Psi(\mathcal{Y})]\Big],$$

$$\mathsf{nd}\Big[\sum_{\mathcal{X}\setminus\mathcal{Z}} \Phi(\mathcal{X})\Big] = \mathsf{nd}\Big[\sum_{\mathcal{X}\setminus\mathcal{Z}} \mathsf{nd}[\Phi(\mathcal{X})]\Big].$$

Those properties guarantee the correctness of computation of non-dominated marginals by *dominance-pruned variable elimination*; that is, by a version of variable elimination in which dominance-pruning is applied after every operation (product or marginalization). If dominance-pruned variable elimination is applied on a multiset $\Gamma$ of set potentials whose joint scopes are $\mathcal{X}$, by the properties above, we have at the end of the computation a real number $r = \mathsf{nd}\big[\sum_{\mathcal{X}} \prod_{\Psi\in\Gamma} \Psi\big]$. Note that while direct computation of the right-hand side of this equality above takes time exponential in the size of the set-potentials in $\Gamma$, applying dominance-pruning after each operation can enormously decrease the overall cost of computation.

The framework thus presented works for LIMIDs with a single value node. From Theorem 1 we know that this is not restrictive in any way. However, it is also possible to extend the framework to directly handle LIMIDs with multiple value nodes, as described by Mauá and de Campos [24]. Shortly, we need to consider pairs of potentials in lieu of potentials and sets of pairs of potentials in lieu of set-potentials. The algebraic operations also need to be slightly modified so as to guarantee the correctness of computations. For the sake of space, we present our algorithms only for the case of univariate utility functions (i.e., LIMIDs with a single value node), but we note that they can be extended to directly deal with multivariate utility functions (i.e., multiple value nodes). Indeed, we compare both approaches in the experiments in Section 6.

### 4.2. Local search with dominance pruning

The main idea here is to use dominance pruning as in the MPU algorithm, but to limit the complexity by performing $k$-local search instead of global search. The crucial step of our proposal is the efficient search for an improving strategy in the $\mathcal{N}$-neighborhood of an incumbent strategy $\delta$, as described in Algorithm 2.

The correctness of the procedure follows immediately from the correctness of set-potential algebra:

**Algorithm 2** Dominance-Based $\mathcal{N}$-Policy Updating (D$\mathcal{N}$PU).

**Require:** A LIMID with a single value node $V$, a subset of the action variables $\mathcal{N}$, an arbitrary strategy $\delta$
  let $\Gamma$ be an initially empty set
  **for all** state variable $S$ **do**
    add a set-potential $\Phi_S = \{P(S|\mathcal{P}_S)\}$ to $\Gamma$
  **end for**
  **for all** action variable $A$ in $\mathcal{N}$ **do**
    add a set-potential $\Phi_A = \{P(A|\delta'_A) : \delta'_A \in \Delta_A\}$ to $\Gamma$
  **end for**
  **for all** action variable $A$ *not* in $\mathcal{N}$ **do**
    add a set-potential $\Phi_A = \{P(A|\delta_A) : \delta_A\}$ to $\Gamma$, where $\delta_A$ is the policy of $A$ in $\delta$
  **end for**
  add the set-potential $\Phi_V = \{U(\mathcal{P}_V)\}$, where $V$ is the value node
  run dominance-pruned variable elimination and return result

**Theorem 3.** *Given a strategy $\delta$, D$\mathcal{N}$PU outputs a strategy $\delta'$ such that $\delta'_A = \delta_A$ for all $A \notin \mathcal{N}$ and $E(U|\delta') \geq E(U|\delta'')$ for all $\mathcal{N}$-neighbor $\delta''$ of $\delta$.*

**Proof.** Let $\Delta(\delta, \mathcal{N})$ be the set of all strategies that agree on $\mathcal{N}$ with $\delta$, that is, all $\mathcal{N}$-neighbors of $\delta$. Note that $\delta$ is an element of $\Delta(\delta, \mathcal{N})$. By construction, we have that

$$\sum_{\mathcal{X}} \prod_{\Psi \in \Gamma} \Psi = \{E(U|\delta') : \delta' \in \Delta(\delta, \mathcal{N})\}.$$

Hence, the result follows from the properties of the algebra of set-potential with dominance pruning. □

If we set $\mathcal{N} = \mathcal{A}$, D$\mathcal{N}$PU collapses to the MPU algorithm [24], and hence produces exact solution of LIMIDs. As with MPU, the worst-case running time of D$\mathcal{N}$PU is exponential in $|\mathcal{N}|$, but dominance pruning can significantly decrease that complexity, as our experiments in Section 6 show.

We call D$k$PU the method that randomly samples a fixed number of sets $\mathcal{N}$ and on each set runs D$\mathcal{N}$PU. Importantly, D1PU offers an algorithm that produces exactly the same result as the popular SPU, but only quicker. As we show in Section 6, the gain in speed is not dramatic for D1PU, but it is very significant for D$k$PU with larger values of $k$. The fact that D$k$PU allows us to try larger values of $k$ in practice is valuable as it leads to superior solutions through local search; depending on the application, even marginal gains can be important, and as such the move from $k$PU to D$k$PU is always recommended.

Note that additional computational savings could be gained by structuring computations in a junction tree (as in SPU), and avoiding redundant computations among different runs of D$k$PU (i.e., with different sets $\mathcal{N}$). We do not study such implementation techniques in this paper.

## 5. Approximate policy updating: A$k$PU

Even though dominance pruning often largely reduces the size of set-potentials, there are cases where pruning is ineffective, as the following example shows.

**Example 1.** Consider a family of LIMIDs (parameterized by a positive integer $n$) with action variables $D_1, \ldots, D_n$, chance variables $A, B, C$, and value variable $V$, connected as in Fig. 3 (for $n = 5$). All variables are binary and take values in $\{0, 1\}$. The CPTs are

$$P(A = 1|D_1, \ldots, D_n) = \sum_i 2^{-i} D_i,$$

$$P(B = 1|D_1, \ldots, D_n) = \sum_i 2^{-i}(1 - D_i),$$

$$\text{and } P(C = 1|A, B) = \begin{cases} 1, & \text{if } A = B = 1 \\ 1/2, & \text{if } A \neq B \\ 0 & \text{if } A = B = 0 \end{cases}.$$

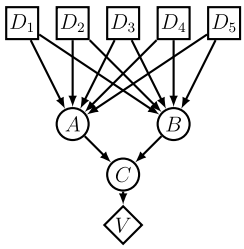


**Fig. 3.** LIMID used in the example where pruning is ineffective.

The utility function is $U(C) = 2^{n+1}C$. Suppose we eliminate variables in order $D_1, \ldots, D_n, C$ and produce

$$\Psi(A, B) = \left\{ \sum_C U(C)P(C|A, B)P(A|d)P(B|d) : d \in \{0, 1\}^n \right\}.$$

Since by construction $P(C = 1|d) = P(A = 1|d)/2 + P(B = 1|d)/2$, we have that

$$\sum_{A,B} \Psi(A, B) = \{2^n P(A = 1|d) + 2^n P(B = 1|d) : d \in \{0, 1\}^n\} = \{2^n - 1\}.$$

Hence, there are $2^n$ non-dominated tables in $\Psi(A, B)$ (if $\phi$ and $\psi$ are two distinct potentials whose values add to the same constant then there are $x$ and $x'$ such that $\phi(x) > \psi(x)$ and $\phi(x') < \psi(x')$).

Bucketing, which we describe next, gives us a way of bounding the growth of tables in such cases at the expense of producing approximate inferences.

Let $M$ be an upper bound we wish to impose over the number of tables in a set-potential during variable elimination. Consider a set-potential $\Phi$ with dimension $d$. Partition the hyper-rectangle $[0, 1]^d$ into a lattice of smaller $M$ hypercubes called buckets. Let $s \stackrel{\text{def}}{=} \lfloor M^{-1/d} \rfloor$. The bucket index of an arbitrary table $\phi = [\phi(x_1), \ldots, \phi(x_d)]$ in $\Phi$ is given by $[\lfloor \phi(x_1)/s \rfloor, \ldots, \lfloor \phi(x_d)/s \rfloor]$. Any two points assigned to the same bucket are less than a distance of $s$ of each other in any coordinate. Thus, by arbitrarily selecting one table in each non-empty bucket we are guaranteed not to introduce a local error of more than $s$. We call this approach A$k$PU. This solution can be improved by any greedy algorithm for clustering under absolute-norm or Euclidean norm (e.g., k-medoids [16]).

**Theorem 4.** *Consider a LIMID of treewidth $w$ and maximum variable cardinality $c$, and let $r$ be the value computed by* A$k$PU *(in fact, with or without dominance pruning) on that LIMID, choosing $M$ at every step in a way that $s \stackrel{\text{def}}{=} \lfloor M^{-1/d} \rfloor$ is bounded from above by a constant $m$, where $d = c^w$. Denoting by $n$ the number of (action, state, and value) variables, we have*

$$\left| r - \max_{\delta} E(U|\delta) \right| \leq 4n^2 \cdot [c + 1]m.$$

**Proof.** Consider set-potentials $\Phi'$ and $\Psi'$ obtained by bucketing of set-potentials $\Phi$ and $\Psi$, respectively. Now consider an element $\gamma = \phi \cdot \psi$ in $\Gamma = \Phi \cdot \Psi$, and let $\gamma' = \phi' \cdot \psi' \in \Gamma' = \Phi' \cdot \Psi'$, where $\phi'$ and $\psi'$ are in the same buckets as $\phi$ and $\psi$, respectively. That is, $|\phi - \phi'| \leq m$ and $|\psi - \psi'| \leq m$. Suppose that $\gamma(x) \geq \gamma'(x)$ at some coordinate $x$. Then

$$\gamma(x) - \gamma'(x) \leq \phi(x)\psi(x) - [\phi(x) + s][\psi(x) - s] = [\phi(x) - \psi(x)] \cdot m - m^2 \leq 2m,$$

where in the last step we assumed that $\phi(x) \geq \psi(x)$ (otherwise $\gamma(x) - \gamma'(x) \leq 0$, contradicting our initial claim) and used Proposition 1 to bound expression $\phi(x) - \psi(x)$ in one. Similarly, suppose that $\gamma'(x) \geq \gamma(x)$. Then

$$\gamma(x) - \gamma'(x) \leq [\phi(x) + s][\psi(x) + s] - \phi(x)\psi(x) = [\phi(x) + \psi(x)] \cdot m + m^2 \leq 3m,$$

where in the last step we used Proposition 1 to bound expression $\phi(x) + \psi(x)$ in two. Hence, for any $\gamma$ in $\Gamma$ there is $\gamma'$ in $\Gamma'$ such that $|\gamma(x) - \gamma'(x)| \leq 3m$. Moreover, if $\Gamma''$ is a set-potential produced by bucketing of $\Gamma'$ that for any $\gamma$ in $\Gamma$ there is $\gamma''$ in $\Gamma''$ such that $|\gamma(x) - \gamma'(x)| \leq 4m$. This implies that bucketing after every product introduces an error of at most $4m$. Consider now a set-potential $\Gamma$ produced by $Y$-marginalization of a set-potential $\Phi(\mathcal{X} \cup \{Y\})$, and let $\Phi'$ be the output of bucketing $\Phi$. For any $\gamma = \sum_Y \phi$ in $\Gamma$ there is $\gamma' = \sum_Y \phi'$ in $\Gamma'$ such that

$$|\gamma(x) - \gamma'(x)| = |\sum_y \phi(x, y) - \sum_y \phi'(x, y)| \leq \sum_y |\phi(x, y) - \phi'(x, y)| \leq c \cdot m.$$

Thus, bucketing after every marginalization introduces an error of at most $[c+1]m$. Variable elimination performs $n-1$ products and $n-1$ marginalizations. Hence the overall error introduced by bucketing is at most $[n-1]\cdot 4\cdot[n-1]\cdot[c+1]m$, and the result follows. □

This result leads to a conservative estimate of the maximum number of buckets $M$ we should use if we want to guarantee before runtime a maximum error on the output. The rationale in the proof above can be used to obtain an estimate of the overall error in the output when we fix the value of $M$ at every step of variable elimination (adjusting it according to the dimension of the tables). We simply need to compute the actual worst-case induced error introduced by bucketing in a given step, accounting for the propagated errors as in the proof: products increase the current error by four, marginalization by $c$. This way, the algorithm can provide bounds on its quality at the end of the computation. When the values of the treewidth and the maximum variable cardinality are bounded by constants, a similar approach serves to prove the existence of an additive fully polynomial-time approximation scheme:

**Theorem 5.** *Given a LIMID of treewidth bounded by a constant $w$ and whose variables have cardinalities bounded by a constant $c$, and $\epsilon > 0$, A$k$PU returns a strategy $\delta_\epsilon$ such that $|E(U|\delta_\epsilon) - \max_\delta E(U|\delta)| \leq \epsilon$ in time polynomial in the size of the input and in $1/\epsilon$.*

**Proof.** Let $n$ be the number of (action, state and value) variables in a LIMID, and $M$ be the maximum number of tables in a set potential produced during a run of variable elimination with dominance pruning on that LIMID. Then MPU takes time $O(c^w \cdot M \cdot n)$, which is $O(M \cdot n)$, as $c^w$ is considered constant. Since bucketing can be computed in time polynomial in $M$ and $c^w$, it follows that A$k$PU takes time polynomial in $M$. Choose $M$ such that $M \geq [4n^2(c+1)]^{c^w}[1/\epsilon]^{c^w} = O(n^\alpha \cdot 1/\epsilon^\beta)$, where $\alpha$ and $\beta$ are some integer constants. Hence, A$k$PU runs in time polynomial in the size of the input (which is at least linear in the number of variables), and in $1/\epsilon$. Let $r$ be the result of A$k$PU and $s \stackrel{\text{def}}{=} \lfloor M^{-1/d} \rfloor$. By Theorem 4, it follows that $|r - \max_\delta E(U|\delta)| \leq \epsilon$. □

For even moderately large values of $n$ or $c$ the estimate $M$ obtained by applying Theorem 4 is prohibitively high, which implies that the above theorem is mostly of theoretical interest except for small diagrams with binary or ternary variables. For example, for a LIMID with structure as in Fig. 2, $n = 100$ and $c = 10$, we have that $m = 1/[4 \cdot 10^5]$. The maximum dimension of a set-potential produced during variable elimination for that LIMID (assuming a perfect elimination order) is $d = 10^3$. Hence, $M \geq s^d \geq 2^{2560}$. A more realistic estimate of the required number of tables at every step can be obtained during runtime by computation of the actual error introduced after every bucketing operation (that is, the maximum discrepancy in each bucket between the tables that were discarded and the one that was kept), and consideration of the propagated error estimates. This way, $M$ can be adjusted adaptively (i.e., increased or decreased according to the current estimate of accumulated errors), potentially demanding much less computational resources than in the proof of Theorem 5, while still guaranteeing a maximum error in the output in fully polynomial time (of course, the worst-case scenario is still given by the bounds in the proof).

## 6. Experiments

We evaluate the algorithms proposed here in a set of randomly sampled LIMIDs with chain-like structure, and in a set of LIMIDs obtained from real-world Bayesian networks. The former allows us to control sources of complexity and better evaluate the performance of the methods with respect to the type of pruning, the width of the search (i.e., the neighborhood size parameter $k$) and the cardinality of variables. The latter test set allows us to analyze the performance of the algorithms on realistic models. It also enables the comparison of two approaches: the transformation of multiple value nodes into a single value node, in which the framework described in Section 4 can be used, or the use of an extended and more complex framework that can accommodate multiple value nodes. The algorithms we consider here are the following:

- $k$PU: (exact) $k$-local search with no pruning and using the framework of propagation of sets of potentials to exploit redundant computations.
- SPU: alias for 1PU (locally optimal).
- D$k$PU: (exact) $k$-local search with dominance pruning.
- A$k$PU: (approximate) $k$-local search with dominance and bucketing pruning.
- MPU: The Multiple Policy Updating algorithm, which is equivalent to one iteration of D$n$PU, where $n$ is the number of action nodes.
- AMPU: Approximate Multiple Policy Updating, which is equivalent to one iteration of A$n$PU, where $n$ is the number of action nodes.

### 6.1. Random chain-like LIMIDs

We start with experiments in randomly sampled LIMIDs with graph structure as in Fig. 2. While the choice of a fixed structure might seem restrictive, we note that any diagram can in principle be transformed into a diagram like that of Fig. 2

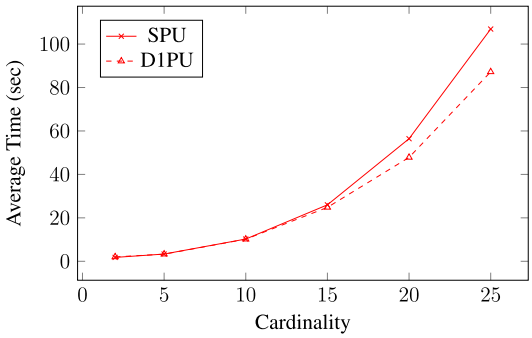


**Fig. 4.** Comparison of average running times of SPU and D1PU on randomly generated chain-like LIMIDs.

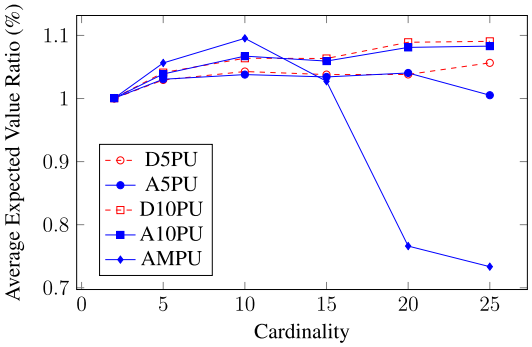


**Fig. 5.** Average accuracy relative to SPU on randomly generated chain-like LIMIDs.

by merging and adding variables [26]. The algorithms were implemented in Python and ran using the Pypy interpreter on desktop computer with a 64 bit intel Xeon 3.4 GHz processor.[1] We performed experiments varying the cardinality $c$ of the variables and the number $n$ of variables. For each configuration of $c$ and $n$, we compared running times and expected value of the best strategy found by the algorithms in a set of 30 LIMIDs, whose conditional probability distributions were independently sampled from a symmetric Dirichlet distribution with parameter $1/c$, and whose utilities were independently sampled from a uniform distribution in $[0, 1]$.

Bucketing was performed with $M = 2^{20}/c^2$, thus keeping the size of set-potentials (number of tables times their dimension) below $2^{20}$, as the treewidth of the LIMIDs we generate is 2. Local search was initialized with a uniform strategy, but while 1-local search was ran until convergence, $k$-local search with $k > 1$ ran for 1000 iterations.

Note first that SPU is by far the most popular algorithm for LIMIDs, and any gain over SPUs speed is welcome. Fig. 4 shows that the overhead of dominance verification pays off in terms of speed. Gains are not dramatic, staying at about 20%, but these gains are obtained without any penalty in the quality of policies (as both SPU and D1PU produce identical results).

Gains in speed with respect to $k$PU are important because they allow one to move up to higher values of $k$, hopefully searching deeper to produce higher expected values. Indeed, experiments summarized by Figs. 5 and 6 show that D$k$PU can be used up to relatively high $k$ with gains in expected value that reach 10%; also, the move to A$k$PU does control computing time while still leading to gains in expected value.

Fig. 5 shows the average of the ratio between expected value obtained with $k$-local search schema to expected value obtained with SPU. Points higher than one indicate that the corresponding method outperforms SPU on average. Dashed curves report the performance of D$k$PU, whereas the solid curves report the performance of A$k$PU. We see that bucketing does not decrease performance significantly, except for AMPU, whose accuracy decays considerably with the increase of variable cardinality. This is likely linked to the number of potentials generated during propagation and to the nature of $k$-search: each iteration of $k$-search generates a much smaller number of potentials than AMPU, hence decreasing the loss introduced by bucketing. Moreover, $k$-search requires only an improving strategy, which minimizes the consequences of pruning good strategies.

Fig. 6 shows average running times. We also see that bucketing adds little overhead to computation for small values of $k$, but that it is crucial for effective tractability for large $k$. Indeed, differently from AMPU, the exact version of MPU was not able to finish computations in most of the diagrams within the limit of one day when $c = 2$. The performance of AMPU is a bit curious: the average runtime increases considerably from $c = 10$ to $c = 15$, then drops significantly from $c = 15$ to

[1] The code is publicly available at http://github.com/denismaua/kpu.

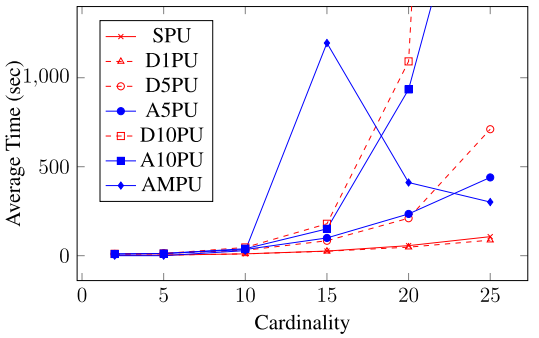


**Fig. 6.** Average time performance on randomly generated chain-like LIMIDs.

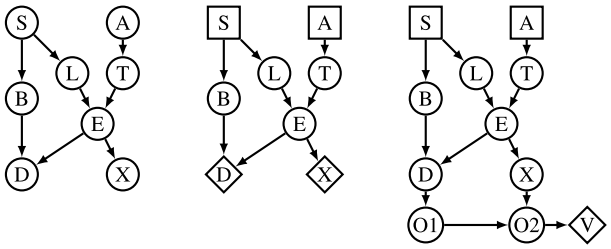


**Fig. 7.** The structure of the Asia Bayesian network and the corresponding LIMIDs generated by procedure described in the text.

$c = 20$, and continues to fall as we increase $c$. This is followed by a similar behavior in accuracy: the average expected value ratio of AMPU is relatively high up to $c = 15$, then falls considerably for $c = 20$ and higher. We conjecture that this is due to dominance pruning being less effective in high dimensional spaces, leading to a great number of potentials falling in the same bucket during propagation. This leads to some arbitrariness in the choices of potentials to discard, but also decreases runtime.

D10PU and A10PU took, respectively, about 9330 and 2600 seconds on average on diagrams with $c = 25$, and are not displayed in the figure for clarity. Although D10PU and A10PU performed on average similarly in time and accuracy, their worst-case running time differed considerably. For example, for $c = 25$, the maximum runtime of D10PU (which took about 10 hours) was one order of magnitude greater than A10PU (which took about one hour).

### *6.2. Realistic diagrams*

The performance of the algorithms varies greatly with the ability of pruning potentials, which in turn is strongly connected to the numerical specification of the diagrams. Hence, it is interesting to study the performance of the proposed methods in diagrams whose probabilities and utilities were elicited from experts (or learned from data) rather than randomly sampled. Unfortunately, realistic LIMIDs are (to our knowledge) not available. To overcome such limitation, we transform Bayesian networks used in real-word applications into LIMIDs as follows.

Given a Bayesian network we transform root nodes into action nodes and leaves into value nodes. The probabilities associated with root nodes are ignored and utility functions for each value node are obtaining by randomly drawing numbers in the range $[0, 1]$ from a (pseudo) uniform distribution. The resulting diagram can be further transformed so as to have a single value variable. Fig. 7 shows the famous Bayesian network asia [7] and the corresponding multivariate utility and univariate utility LIMIDs. To minimize the arbitrariness introduced by artificially generating utility functions, we repeat the procedure above 30 times, hence generating for each Bayesian network 30 LIMIDs with potentially many value nodes and 30 LIMIDs with a single value node.

The motivation to generate LIMIDs structured as described above comes from practical applications of LIMIDs in decision support systems for military operations, viral marketing and fault analysis [8,14,28]. In such domains, knowledge is usually obtained by a process of either forward or backward reasoning, in which one starts with an event of interest, and recursively attempts at determining either its possible effects or causes. The goal is to determine which interventions are most cost-effective in producing or avoiding a certain event. For instance, in planning Effects-Based Operations (EBO), possible military options such as planning an air strike or sending ground troops correspond to action nodes with children denoting events such as eliminating radars and conquering enemy space, which in turn may have value nodes as children representing the number of casualties, machinery damage and victory [8]. The goal is to select which actions to take such as to balance the chance of victory and the financial and human losses.

**Table 1**
Bayesian networks used to generate realistic MEU problems. N, R and L denote, resp., the number of nodes, root nodes and leaves in each network. TW is an estimate of the network's treewidth. $\lg|\Delta|$ is the log of the search space size of the corresponding MEU problem.

| Model | N | R | L | TW | $\lg\|\Delta\|$ | Domain | Ref. |
|---|---|---|---|---|---|---|---|
| Alarm | 37 | 12 | 11 | 4 | 13 | Health Care | [3] |
| Mildew | 36 | 16 | 1 | 4 | 32 | Agriculture | |
| Hailfinder | 69 | 17 | 13 | 4 | 32 | Weather Forecasting | [1] |
| Win95pts | 92 | 34 | 16 | 8 | 34 | Computer Troobleshooting | [4] |
| Andes | 248 | 89 | 25 | 17 | 89 | Education | [5] |
| Diabetes | 415 | 76 | 2 | 4 | 269 | Health Care | [2] |

We selected a group of Bayesian networks of varying complexity used in real-word applications and publicly available on the Internet.[2] Table 1 shows relevant characteristics of the models used to generate LIMIDs. The search space size ($\lg|\Delta|$) and the treewidth provide a measure of the difficulty of solving each model by a brute-force approach, that is, by enumerating and evaluating all strategies. For instance, the diagrams generated from the Alarm network require $2^{13} = 8192$ strategy evaluations on diagram of treewidth between 4 and 7 (recall that the transformation from multiple-value variable diagrams into single-value variable diagrams may increase treewidth by at most three, and that exact strategy evaluation takes time exponential on the treewidth). On the other hand, diagrams generated from the Win95pts network would require $2^{34} > 10^9$ strategy evaluations on diagrams of treewidth between 8 and 11. Thus, while the former kind of problems can easily be solved by brute-force in current desktop computers in a reasonable amount of time, solving the latter type of problems by enumeration would take a considerable amount of time. From now on, we refer to the set of diagrams generated from a certain Bayesian network by the same name; for instance, we say that Alarm has 12 action nodes, meaning that all diagrams in the set of LIMIDs generated by the procedure described using the Alarm networks as input have 12 action nodes.

To speed up things, we re-implemented all algorithms in C++. For the following experiments, we allowed each problem instance to consume at most 1 h of CPU time and 12 GB of memory (and we killed instances taking more resources than allowed, considering them *unsolved*).

*6.2.1. Multivariate vs. univariate utilities*

As discussed in Section 4, the framework on which both D*k*PU and A*k*PU are based handle only univariate utility functions (i.e., diagrams with a single value variable). By Theorem 1 we know that any diagram can efficiently be transformed to meet such a condition. The transformation however increases treewidth and inserts new variables which might change the ability to prune potentials. An alternative method is to extend the framework to accommodate multivariate utility functions, as in the work of Mauá and de Campos [24]. To do so requires working with pairs of potentials, which introduces a small overhead in the propagation of potentials, and also leads to a potential decrease in the number of pruned potentials, due to the increase in the local dimension.

We evaluated the effect of either approach (i.e., converting into single-value variable diagrams versus extending the framework of set-potentials) in the performance of A*k*PU in the test set of realistic diagrams. The average running time (in seconds) and accuracy grouped by model and search width are shown in Table 2. Note that we omit the results on influence diagrams generated with the Mildew network since it contains a single leaf node (hence it produces diagrams with a single value node).

The results show that for nearly all cases, the conversion of diagrams with multivariate utility into diagrams with univariate utility creates MEU problems on which A*k*PU performs worse both in speed and accuracy. As discussed, this might be caused by the increase in treewidth; a second reason is that the transformation might lead to graphical structures which are harder for heuristics that compute elimination sequences (necessary to evaluate strategies); in our experiments we obtain such sequences by applying the well-known min-fill heuristic. The increase in complexity is often mild, especially for small search width: in Alarm diagrams A*k*PU was on average two to three to 17 times slower in the univariate utility diagrams for $k = 1, 5, 10$, respectively. In that set AMPU was over 300 times faster in the multivariate utility diagrams. In Win95pts diagrams the speed gain of A*k*PU in multivariate utility diagrams was smaller but still noticeable, while AMPU was significantly faster in multivariate utility diagrams. In the Diabetes diagrams, the algorithms performed slightly better on the univariate utility diagrams. The most striking difference is observed on the diagrams generated from the Andes network; for instance, A5PU ran on average about 88 times slower on univariate utility diagrams, and while A10PU took on average 61.77 seconds to finish in multivariate utility diagrams, it did not finish within 1 h in any diagram.

Regarding the quality of the strategies found, the algorithm performed very similarly on both types of diagrams. In the Alarm diagrams, results on the multivariate utility diagrams show higher average expected utility for $k = 1, 5$ and similar value for the other values of $k$. In Hailfinder, the strategies found by A*k*PU had on average higher values in multivariate utility diagrams when $k = 1$ or $k = 5$, and smaller values when $k = 10$; the latter was true also for AMPU. In Win95pts, the use of multivariate diagrams led to better strategies for A*k*PU (i.e., higher values) but worse strategies (i.e., smaller values)

[2] The networks used were obtained at http://www.cs.huji.ac.il/site/labs/compbio/Repository and http://www.bnlearn.com/bnrepository/.

**Table 2**
Comparison of multivariate and univariate utility approaches on realistic diagrams. The symbol – denotes instances which the corresponding method was unable to solve within the given resources. Results followed with a number inside parenthesis indicate cases where only part of the instances was solved (viz. the number inside the parenthesis).

| Model | Width | Multivariate | | Univariate | |
|---|---|---|---|---|---|
| | | Time | Value | Time | Value |
| Alarm | 1 | 0.013 | 6.668744 | 0.037 | 6.661484 |
| | 5 | 0.021 | 6.744249 | 0.072 | 6.708198 |
| | 10 | 0.038 | 6.779323 | 0.653 | 6.779323 |
| | All | 0.004 | 6.779323 | 0.294 | 6.779323 |
| Hailfinder | 1 | 0.06 | 6.621059 | 0.06 | 6.620711 |
| | 5 | 0.64 | 6.621714 | 0.48 | 6.621601 |
| | 10 | 487.52 | 6.621208 | 1127.57(15) | 6.742567(15) |
| | All | 27.41 | 6.557055 | 39.83 | 6.575807 |
| Win95pts | 1 | 0.10 | 9.597652 | 0.17 | 9.554800 |
| | 5 | 0.16 | 9.651433 | 0.29 | 9.568741 |
| | 10 | 0.17 | 9.669896 | 0.44 | 9.623285 |
| | All | 14.11 | 9.681265 | 166.18 | 9.718513 |
| Andes | 1 | 16.21 | 14.935538 | 1627.74 | 14.962035 |
| | 5 | 25.45 | 14.935538 | 2199.16(24) | 14.931065(24) |
| | 10 | 61.77 | 14.954029 | – | – |
| | All | – | – | – | – |
| Diabetes | 1 | 406.04 | 1.466960 | 334.78 | 1.456103 |
| | 5 | – | – | – | – |
| | 10 | – | – | – | – |
| | All | – | – | – | – |

for AMPU. In Diabetes, the strategies found by A1PU were on average slightly superior in multivariate utility diagrams. Finally, in Andes, A1PU obtained slightly better strategies on univariate utility diagrams, and A5PU produced slightly worse strategies on multivariate utility diagrams.

It is worth noting that A$k$PU is outperformed by AMPU in terms of speed and accuracy in the multivariate utility diagrams of the Alarm set for any value of $k$; this relates well to the hardness of brute-force solving those instances, as discussed previously. AMPU also outperformed A10PU in Hailfinder diagrams (but not A1PU and A5PU). This shows that dominance pruning is quite effective in many diagrams, making $k$-local search compares unfavorably in those cases due to the time overhead introduced by solving multiple $k$PI. At the same time, the results show that A$k$PU can lead to significant time savings with respect to AMPU when dominance pruning is ineffective, for instance in the univariate utility diagrams generated with Win95pts. We also note that in the Andes set, AMPU was not able to finish within the allowed time in any instance, whereas A10PU solved problems on average within slightly more than 1 minute. A5PU, A10PU and AMPU could not solve any instance of the Diabetes diagrams with the allowed resources (memory was usually the bottleneck in those cases).

Overall, the results in this section suggest that extending the framework to deal with multivariate utility functions is beneficial to both accuracy and speed performance of A$k$PU and AMPU algorithms.

*6.2.2. The effect of (approximate) pruning*

As discussed, the variants of A$k$PU and AMPU that directly handle multivariate utility perform better on average than their counterparts that require reducing problems to single value variable diagrams. This is also true for $k$PU (i.e., for $k$-local search with no pruning), as the transformation from diagrams with multiple value nodes into diagrams with a single value node most often increases treewidth and reduces the effectiveness of dominance pruning. Hence, in this section, we consider only the multivariate utility diagrams of the corresponding sets, and we evaluate the effect of dominance and bucketing pruning in $k$-local search in both accuracy and speed.

Table 3 shows the average running time and accuracy of A$k$PU and $k$PU in LIMIDs generated from Bayesian networks. We see that overall pruning strategies either show virtually no difference in speed or slow down search when $k = 1$; for $k = 5$ there is a minor reduction in running time in Alarm and Win95pts diagrams, a slightly bigger reduction in Andes diagrams and significant reduction in Hailfinder and Mildew diagrams. The situation is more complex for $k = 10$. A10PU was over 3 times faster than 10PU in Alarm diagrams. On average, the maximum number of potentials generated by 10PU over Hailfinder diagrams was 663 552 against an average of 64 800 potentials of A10PU; yet, the latter was almost two times slower in Hailfinder diagrams. A10PU was almost 5 times faster in Win95pts diagrams and approximately 1.2 faster in Andes diagrams. While A10PU took an average of 14.11 seconds to solve the diagrams generated from the Win95pts network, 10PU did not solve any instance. No method was able to solve diagrams generated with the Mildew network for $k10$ with the given resources, and for $k = 5$ or higher with the Diabetes network.

We also see that approximate pruning incurs a negligible loss in accuracy. The most discrepant results are observed in Alarm diagrams, where A5PU obtains 0.023 higher expected utility on average, and in Win95PTs diagrams, where A5PU

**Table 3**
Comparison of pruned and non-pruned variants of *k*PU on realistic diagrams.

| Model | Width | A*k*PU | | *k*PU | |
|---|---|---|---|---|---|
| | | Time | Value | Time | Value |
| Alarm | 1 | 0.013 | 6.668744 | 0.012 | 6.668744 |
| | 5 | 0.021 | 6.744249 | 0.024 | 6.767229 |
| | 10 | 0.038 | 6.779323 | 0.121 | 6.779323 |
| | All | 0.004 | 6.779323 | 0.020 | 6.779323 |
| Mildew | 1 | 13.83 | 0.513724 | 13.69 | 0.513724 |
| | 5 | 145.09 | 0.513913 | 264.51 | 0.513798 |
| | 10 | – | – | – | |
| Hailfinder | 1 | 0.06 | 6.621059 | 0.06 | 6.621059 |
| | 5 | 0.64 | 6.621714 | 1.01 | 6.621088 |
| | 10 | 487.52 | 6.621208 | 258.91 | 6.621701 |
| | All | 27.41 | 6.557055 | – | – |
| Win95pts | 1 | 0.10 | 9.597652 | 0.10 | 9.597652 |
| | 5 | 0.16 | 9.651433 | 0.17 | 9.598018 |
| | 10 | 0.17 | 9.669896 | 0.82 | 9.629361 |
| | All | 14.11 | 9.681265 | – | – |
| Andes | 1 | 16.21 | 14.935538 | 15.61 | 14.935538 |
| | 5 | 25.45 | 14.935538 | 30.73 | 14.935538 |
| | 10 | 61.77 | 14.954029 | 152.95 | 14.941875 |
| Diabetes | 1 | 406.04 | 1.466960 | 411.95 | 1.466961 |
| | 5 | – | – | – | – |
| | 10 | – | – | – | – |
| | All | – | – | – | – |

and A10PU obtains respectively 0.04 and 0.05 gains. We note that the gains produced by larger search widths (i.e., by large values of $k$) are minor, but usually come at little cost. A10PU was less than four times slower than A1PU in Alarm, Win95pts and Andes diagrams. On the other hand, A10PU was 8000 times slower than A1PU in Hailfinder diagrams and unable to solve instances with the given resources in Mildew diagrams. Interestingly, pruning lead to more accurate results for $k = 5$ and $k = 10$ in Win95pts diagrams; this can be explained by the ability of pruning to select locally suboptimal solutions, which might help escaping local optimal and ultimately lead to higher expected utility strategies.

Most of the action variables in the realistic diagrams take on between two and four values. Hence, one possible cause for the slightly worse performance of A1PU relative to 1PU is that most of the 1PI problems generated during the 1-local search were too simple, causing pruning to be ineffective. A similar phenomenon might have occurred with $k = 5$: most of the *k*PI problems generated had very small search space and the gains from pruning were negligible. It is possible to enable pruning only for those searches where the search space is significantly large. However, predicting when pruning will be effective is challenging and demands special care; we leave this possibility for future work.

All in all, the results in this section suggest that pruning speeds up $k$-local search, especially when $k$ is large; the only counterexample are the diagrams generated using the Hailfinder network for $k = 10$ (in which case running AMPU was more efficient though less accurate). We note that $k$-local search naturally offers an anytime scheme: start with $k = 1$ and gradually increase $k$. The results reported here show that increase in running time of A*k*PU is often linear in the width, making it an effective strategy.

## 7. Conclusion

Limited memory influence diagrams (LIMID) offer a rich graphical language to describe decision problems, allowing the representation of limited information scenarios which often arises in real applications. Finding an optimal strategy for a LIMID is an NP-hard problem, and practitioners resort to local search algorithms. For instance, the popular SPU algorithm implements 1-neighborhood local search.

In this paper we investigated means of speeding up local search algorithms. We showed that $k$-local search is W[1]-hard, and hence unlikely to be polynomial-time tractable. We also showed that obtaining strategies that resemble optimal ones (even though their expected utility is low) is NP-hard. We then developed fast local search algorithms based on dominance pruning. Even with dominance pruning, searching for a $k$-policy improvement for moderately large values of $k$ can be slow. To remedy this, we designed an approximate pruning strategy that removes a strategy if there is another close enough strategy (in terms of L1-norm). We proved that the approximate pruning leads to a fully polynomial additive approximation algorithm in bounded-treewidth bounded-variable cardinality LIMIDs if we set $k$ to be the number of action variables.

Experiments with randomly generated regular diagrams of bounded treewidth and varying variable cardinality showed that dominance pruning speeds up computations even for 1-neighborhood local search (i.e., SPU). Moreover, dominance pruning becomes essential for $k$-local search with $k > 5$. We also empirically showed that the quality of approximate pruning

decays quickly with the increase of variable cardinalities, being only useful for variable with cardinalities up to 15 or so. We observed a slightly different situation in diagrams derived from real-world Bayesian networks. In those diagrams, (approximate) pruning was ineffective or even harmful when $k = 1$. On the other hand it led to drastic gains in efficiency when $k = 10$ in most diagrams, and already for $k = 5$ in some. We also showed that approximate pruning introduces only minor losses in accuracy. Importantly, approximate pruning offers a way to trade off accuracy and speed, beyond the natural trade off enabled by the search width of $k$-local search approaches.

## Acknowledgements

The first author was partially supported by the São Paulo Research Foundation (FAPESP) Grant No. 13/23197-4. The second author is partially supported by CNPq.

## References

[1] B. Abramson, J. Brown, W. Edwards, A. Murphy, R.L. Winkler, Hailfinder: a Bayesian system for forecasting severe weather, Int. J. Forecast. 12 (1) (1996) 57–71.
[2] S. Andreassen, R. Hovorka, J. Benn, K.G. Olesen, E.R. Carson, A model-based approach to insulin adjustment, in: Proceedings of the 3rd Conference on Artificial Intelligence in Medicine, 1991, pp. 239–248.
[3] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, G.F. Cooper, The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks, 1989.
[4] J.S. Breese, D. Heckerman, Decision-theoretic troubleshooting: a framework for repair and experiment, in: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI), 1996, pp. 124–132.
[5] C. Conati, A.S. Gertner, K. VanLehn, M.J. Druzdzel, On-line student modeling for coached problem solving using Bayesian networks, in: Proceedings of the 6th International Conference on User Modeling, 1997, pp. 231–242.
[6] G.F. Cooper, A method for using belief networks as influence diagrams, in: Fourth Workshop on Uncertainty in Artificial Intelligence, 1988.
[7] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, D.J. Spiegelhalter, Probabilistic Networks and Expert Systems, Springer, 1999.
[8] C.P. de Campos, Q. Ji, Strategy selection in influence diagrams using imprecise probabilities, in: Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI), 2008, pp. 121–128.
[9] R. Dechter, An anytime approximation for optimizing policies under uncertainty, in: Workshop of Decision Theoretic Planning, AIPS, 2000.
[10] A. Detwarasiti, R.D. Shachter, Influence diagrams for team decision analysis, Decis. Anal. 2 (4) (2005) 207–228.
[11] R. Downey, M.R. Fellows, Fixed-parameter tractability and completeness I: basic theory, SIAM J. Comput. 24 (1995) 873–921.
[12] R. Downey, M.R. Fellows, Fixed-parameter tractability and completeness II: completeness for W[1], Theor. Comput. Sci. A 141 (1995) 109–131.
[13] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer-Verlag, 1999.
[14] B.C. Ezell, S.P. Bennett, D. von Winterfeldt, J. Sokolowski, A.J. Collins, Probabilistic risk analysis and terrorism risk, Risk Anal. 30 (4) (2010) 575–589.
[15] U. Feige, M. Langberg, K. Nissim, On the hardness of approximating np witnesses, in: Klaus Jansen, Samir Khuller (Eds.), Approximation Algorithms for Combinatorial Optimization, in: Lecture Notes in Computer Science, vol. 1913, Springer, Berlin, Heidelberg, 2000, pp. 120–131.
[16] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd edn., Springer, 2009.
[17] R.A. Howard, J.E. Matheson, Influence diagrams, in: Readings on the Principles and Applications of Decision Analysis, Strategic Decisions Group, 1984, pp. 721–762.
[18] F.V. Jensen, T.D. Nielsen, Bayesian Networks and Decision Graphs, 2nd edition, Information Science and Statistics, Springer, 2007.
[19] A. Khaled, C. Yuan, E. Hansen, Solving limited memory influence diagrams using branch-and-bound search, in: Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI), 2013.
[20] J. Kohlas, Information Algebras: Generic Structures for Inference, Springer-Verlag, New York, USA, 2003.
[21] J.H.P. Kwisthout, H.L. Bodlaender, L.C. van der Gaag, The necessity of bounded treewidth for efficient inference in Bayesian networks, in: Proceedings of the 19th European Conference on Artificial Intelligence, 2010, pp. 237–242.
[22] S.L. Lauritzen, D. Nilsson, Representing and solving decision problems with limited information, Manag. Sci. 47 (2001) 1235–1251.
[23] Q. Liu, A. Ihler, Belief propagation for structured decision making, in: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI), 2012, pp. 523–532.
[24] D.D. Mauá, C.P. de Campos, Solving decision problems with limited information, in: Advances in Neural Information Processing Systems (NIPS), vol. 24, 2011, pp. 603–611.
[25] D.D. Mauá, C.P. de Campos, M. Zaffalon, The complexity of approximately solving influence diagrams, in: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI), 2012, pp. 604–613.
[26] D.D. Mauá, C.P. de Campos, M. Zaffalon, Solving limited memory influence diagrams, J. Artif. Intell. Res. 44 (2012) 97–140.
[27] D.D. Mauá, C.P. de Campos, Marco Zaffalon, On the complexity of solving polytree-shaped limited memory influence diagrams with binary variables, Artif. Intell. 205 (2013) 30–38.
[28] A. Nath, P. Domingos, Efficient belief propagation for utility maximization and repeated inference, in: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.
[29] J.D. Park, A. Darwiche, Complexity results and approximation strategies for MAP explanations, J. Artif. Intell. Res. 21 (2004) 101–133.
[30] D. Roth, On the hardness of approximate reasoning, Artif. Intell. 82 (1–2) (1996) 273–302.
[31] S. Szeider, The parameterized complexity of k-flip local search for SAT and MAX SAT, Discrete Optim. 8 (1) (2011) 139–145.
[32] J.A. Tatman, R.D. Shachter, Dynamic programming and influence diagrams, IEEE Trans. Syst. Man Cybern. 20 (2) (1990) 365–379.
[33] W. Watthayu, Representing and solving influence diagram in multi-criteria decision making: a loopy belief propagation method, in: Proceedings of the International Symposium on Computer Science and Its Applications, 2008, pp. 118–125.
[34] N.L. Zhang, R. Qi, D. Poole, A computational theory of decision networks, Int. J. Approx. Reason. 11 (2) (1994) 83–158.