Speeding Up *k*-Neighborhood Local Search in Limited Memory Influence Diagrams

Denis Deratani Mauá and Fabio Gagliardi Cozman

Universidade de São Paulo,
São Paulo, Brazil
{denis.maua,fgcozman}@usp.br

Abstract. Limited memory influence diagrams are graph-based models that describe decision problems with limited information, as in the case of teams and agents with imperfect recall. Solving a (limited memory) influence diagram is an NP-hard problem, often approached through local search. In this paper we investigate algorithms for k-neighborhood local search. We show that finding a k-neighbor that improves on the current solution is W[1]-hard and hence unlikely to be polynomial-time tractable. 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 particularly suited for teams and limited-resource agents [4,13]. LIMIDs relax the *perfect recall* requirement (a.k.a. no-forgetting assumption) of traditional influence diagrams [9], 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 [18], or if variables are binary and the utility function is multivariate [16]. Similar negative results hold for approximating the problem within any fixed constant when either variable cardinality or treewidth is unbounded [18]. And even though there are polynomial-time approximations when cardinalities and treewidth are bounded [17], 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) [15], 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 [3,2,10].

In practice, local search methods are the most widely used algorithms for approximately solving LIMIDs. Lauritzen and Nilsson [13] developed the Single Policy Updating (SPU) algorithm for computing locally optimum strategies in arbitrary LIMIDs. Their algorithm remains the most referenced and probably used algorithm for solving medium and large LIMIDs. SPU iteratively seeks

L.C. van der Gaag and A.J. Feelders (Eds.): PGM 2014, LNAI 8754, pp. 334–349, 2014.

for a variable that can improve the incumbent global strategy by modifying its associated actions. If no such variable is found the algorithm halts at a local optimum. Detwarasiti and Shachter [4] 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 every step requires exploration of $O(n^k)$ candidates. There have also been proposals based on message-passing algorithms to cope with high treewidth diagrams [14].

In this paper we focus on the local search that runs within the methods in the last paragraph. First, we prove that k-local search is W[1]-hard, which suggests that algorithms that run in time $O(f(k)n^{\alpha})$ are unlikely to exist, where f is an arbitrary computable function and α is a constant independent of k (Section 3). We take such a result as an indication that approximating local-search is necessary for large values of k.

We thus investigate the use of MPU's pruning to speed up SPU and related k-local search schema (Section 4). We propose a relaxed and approximate version of MPU's 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 networks 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 [8]. 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 assess the quality of decisions for every configuration of state variables. 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 [22], 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 δ_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 Δ_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 not practicable.

The *perfect recall* condition (a.k.a. no forgetting) assumes that all decisions and observations are "remembered". Graphically, it entails that if A and A' are two action nodes such that A is a parent of A', then all parents of A are also parents of A'. We assume that when perfect recall is satisfied the "remembered" arcs are explicitly represented in the diagram.

The construction of an optimal strategy is harder for LIMIDs that do not satisfy the perfect recall requirement, exactly due to the absence of links between actions.

Given an action variable A and a policy δ_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 δ_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 δ 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 δ, $E(U|\delta)$, is then $\sum_{\mathcal{S},\mathcal{A}} U(\mathcal{S}, \mathcal{A}) P(\mathcal{S}, \mathcal{A}|\delta)$.

Given a strategy δ, computing $E(U|\delta)$ can be reduced to a marginal inference in a Bayesian network [1]. Conversely, marginal inference in Bayesian networks, a $\#P$-complete problem [20], can be reduced 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 for networks of small treewidth (with treewidth considered constant in the complexity analysis). Kwisthout et al. [12] 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).

An important task with LIMIDs is that of finding the

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

MEU is $\mathrm{NP}^{\mathrm{PP}}$-complete, and NP-complete for diagrams of bounded treewidth [2]. The problem is NP-complete on LIMIDs of bounded treewidth even when all variables are binary [16], and when all variables are ternary and there is a single value node [18].

Provided that expected utilities can be succinctly encoded (i.e., represented in space $O(b^{\alpha})$, where b is the size of the encoding and α is a constant), we can compute the value of the maximum expected utility by binary search in polynomial time if MEU can be solved in polynomial time. Similarly, if the in-degrees of action nodes are bounded we can use a polynomial-time algorithm M that solves MEU to obtain an optimal strategy in polynomial time as follows. First, perform a binary search using M to compute the maximum expected utility and use that value as k. Select an action variable A and for every policy δ_A build a new LIMID where A is a state node with conditional probability $P(A|\delta_A)$. Now run the algorithm M on those LIMIDs: the algorithm will certainly return a yes answer on some of them; any policy δ_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, MEU can be efficiently solved by any polynomial-time algorithm that computes the maximum expected utility. Finally, if the treewidth of the diagrams is bounded, MEU can trivially be solved in polynomial-time by any polynomial-time algorithm that finds optimal strategies. Thus, MEU 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 in [17] and [18].

Proposition 1. *Given a LIMID $\mathcal{L}$ of treewidth w we can construct in time polynomial in its size a LIMID $\mathcal{L}'$ and a 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 δ' of $\mathcal{L}'$ into strategies δ of $\mathcal{L}$ in linear time; (iv) if δ' is such that $E(U'|\delta') > 0$ and $\delta = f(\delta')$ then $E(U|\delta) \propto E(U'|\delta')$, where U and U' denote the utility functions of $\mathcal{L}$ and $\mathcal{L}'$, respectively.*

A corollary of the above result is that the MEU of $\mathcal{L}'$ equals the MEU 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 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.

3 The Complexity of k-Neighborhood Local Search

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

> **k-Policy Updating (kPU)** Take a LIMID, a strategy δ_0 and a positive integer M: for $i = 1 \dots M$ find a strategy δ_i in the k-neighborhood of δ_{i-1}; at the end, return δ_M.

For large enough and finite M the procedure converges to a local optimum. In particular, if k equals the number of action variables, a global optimum is found in one iteration. The main bottleneck of kPU is the k-local search step, where an improving solution is searched for; this can be formalized as

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

The same argument used when discussing MEU 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 kPI in the sense that (under the same requirements) a polynomial-time algorithm for one problem can be used to solve another.

For a LIMID whose action variables are parentless, kPI 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 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 α 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 in the rich field of parameterized complexity.

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 running time 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 α is a constant that does not depend on k [5]. 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 [6]: if FPT=W[1] then NP-complete problems can be solved in subexponential time [7].

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 α is a constant.

The next result shows that if a fixed-parameter tractable algorithm that performs k-local search on the space of strategies existed we would prove k-FLIP MAX SAT to be fixed-parameter tractable.

Theorem 1. *Unless W[1]=FPT, there is no algorithm that solves k-POLICY IMPROVEMENT in time $O(f(k)b^{\alpha})$, where b is the size of bitstring encoding of the LIMID, f is an arbitrary computable function and α 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 τ_0
Parameter: A positive integer k
Question: Is there a k-flip of τ_0 satisfying more clauses of F?

A k-flip of an assignment τ is another assignment that differs from τ in the values assigned to at most k variables. Szeider [21] showed that the above problem is W[1]-hard.

Consider formula F, truth assignment τ 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 Figure 1 and numerical parameters specified as follows (this construction is similar to the construction used by [19] 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:

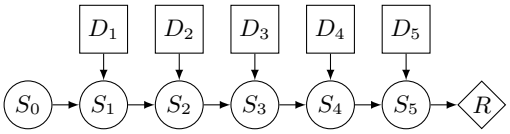

Fig. 1. LIMID used to prove Theorem 1

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

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 $\#\mathsf{SAT}$ is the number of clauses satisfied by a truth-value assignment corresponding to δ. Consider an arbitrary strategy δ corresponding to the truth-value assignment τ. A k-neighbor of δ is a strategy δ' differing from δ in at most k coordinates. Hence, there is a truth-value assignment satisfying more clauses in F than τ if and only there is k-policy improvement of δ, and the result is proved. □

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 allows us to climb up to reasonably large k (say, 10) for large values of n.

Assuming (w.l.o.g.) that a LIMID has parentless action variables of cardinality c, a brute-force approach to k-local search can be accomplished by examining the 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 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 ϕ *with* scope $\mathcal{X}$ *is a nonnegative real-valued mapping of configurations* x *of* $\mathcal{X}$.

We assume the usual algebra of potentials: the product $\phi \cdot \psi$ of potentials ϕ and ψ returns a potential γ 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 ϕ with scope $\mathcal{X}$, where $\mathcal{Y} \subseteq X$, is the potential ψ 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 [11]. This entails that 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 Γ all potentials whose scope contain X, compute the marginal of those products which sums over X and add the result to Γ.

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 2. *Consider a LIMID with a single value variable* V *and a strategy* δ*, 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* ϕ *generated during variable elimination satisfies* $0 \leq \phi \leq 1$.

Proof. Consider the step in variable elimination where all potentials containing a variable X are collected, and let $\mathcal{Y}$ be all variables $Y < X$. Let also F_Y be the set of Y and its children. By design, the only potentials in the initial Γ 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 $\psi_X = \sum_X P(X|\mathcal{P}_X) \sum_{\mathcal{Y}} \prod_{Z\in F_Y : Z\neq X, Y\mathcal{Y}} P(Z|\mathcal{P}_Z)$, 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 MEU [15]. To do so, we define:

Definition 2. *A* set-potential $\Phi(\mathcal{X})$ *is a set of potentials* ϕ *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* $[\sum_{\mathcal{X}\setminus\mathcal{Y}} \Phi](\mathcal{Y}) = \{\sum_{\mathcal{X}\setminus\mathcal{Y}} \phi : \phi \in \Phi\}$.

Mauá et al. [18] proved that the algebra of set-potentials is a valuation algebra [11], and thus marginal inference with set-potentials can also be computed by variable elimination (with potentials and their operations replaced by their set counterpart).

The product of set-potentials may create exponentially larger 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 ϕ dominates (resp., is dominated by) ψ 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 Φ.*

Note that if nd is applied on a set-potential with empty scope, it produces a single real number. Mauá et al. [18] showed that dominance-pruning satisfies

$$\mathsf{nd}[\Phi(\mathcal{X})\Psi(\mathcal{Y})] = \mathsf{nd}\left[\mathsf{nd}[\Phi(\mathcal{X})]\mathsf{nd}[\Psi(\mathcal{Y})]\right], \quad \mathsf{nd}\left[\sum_{\mathcal{X}\setminus\mathcal{Z}} \Phi(\mathcal{X})\right] = \mathsf{nd}\left[\sum_{\mathcal{X}\setminus\mathcal{Z}} \mathsf{nd}[\Phi(\mathcal{X})]\right].$$

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 multi-set Γ of set potentials whose joint scopes are $\mathcal{X}$, by the properties above, we have at the end of the computation a real number $r = \mathsf{nd}\left[\sum_{\mathcal{X}} \prod_{\Psi\in\Gamma} \Psi\right]$. 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 Γ, applying dominance-pruning after each operation can enormously decrease the overall cost of computation.

4.2 Local Search With Dominance Pruning

The main idea here is to use dominance pruning as in the MPU algorithm, but to run efficient k-local search with the space of strategies. Our proposal is as follows.

Dominance-Based $\mathcal{N}$-Policy Updating (D$\mathcal{N}$PU). Let $\mathcal{N}$ be a subset of the action variables, δ be an arbitrary strategy, and Γ be an initially empty set.

1. For each state variable S add a set-potential $\Phi_S = \{P(S|\mathcal{P}_S)\}$ to Γ,
2. for each action variable A in $\mathcal{N}$ add a set-potential $\Phi_A = \{P(A|\delta'_A) : \delta'_A \in \Delta_A\}$ to Γ,
3. for each action variable A *not* in $\mathcal{N}$ add a set-potential $\Phi_A = \{P(A|\delta_A) : \delta_A\}$ to Γ, where δ_A is the policy of A in δ.

4. Add the set-potential $\Phi_V = \{U(\mathcal{P}_V)\}$, where V is the value node,
5. run dominance-pruned variable elimination and return result.

Theorem 2. *D$\mathcal{N}$PU outputs a strategy δ' such that $\delta'_A = \delta_A$ for all $A \notin \mathcal{N}$ and $E(U|\delta') \geq E(U|\delta)$.*

Proof. Let $\Delta(\delta, \mathcal{N})$ be the set of all strategies that agree on $\mathcal{N}$ with δ, that is, all $\mathcal{N}$-neighbors of δ. Note that δ is an element of $\Delta(\delta, \mathcal{N})$. By design, 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 [15], and hence produces exact solution of LIMIDs (of moderate size). 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 DkPU the method that randomly samples a fixed number of sets $\mathcal{N}$ and on each set run 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 DkPU with larger values of k. The fact that DkPU 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 kPU to DkPU 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 DkPU (i.e., with different sets $\mathcal{N}$). We do not study such implementation techniques in this paper.

5 Approximate Policy Updating: AkPU

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 LIMID with action variables $D_1, \ldots, D_n$, state variables A, B, C and utility node V. A and B have either all action variables as parents. C has A and B as parents and V as child. All variables are binary and take values in $\{0, 1\}$. The CPTs are $P(A{=}1|D_1, \ldots, D_n) = \sum_i 2^{-i} D_i$, $P(B{=}1|D_1, \ldots, D_n) = \sum_i 2^{-i}(1 - D_i)$, and $P(C = 1|A, B)$ is 1 if $A = B = 1$, 1/2 if $A \neq B$, and 0 if $A{=}B{=}0$. The utility is $U(C) = 2^{n+1}C$. Suppose we eliminate variables in order $D_1, \ldots, D_n, C$ and produce $\Psi(A, B) = \{\sum_C U(C)P(C|A, B)P(A|d)P(B|d) : d \in \{0, 1\}^n\}$. We have that $\sum_{A,B} \Psi(A, B) = \{2^n P(A|d) + 2^n P(B|d) : d \in \{0, 1\}^n\} = \{2^n - 1\}$. Hence, there are 2^n non-dominated tables in $\Psi(A, B)$ (as two potentials whose values add to the same constant cannot one dominate each other).

Bucketing, which we describe next, gives us a way of bounding the growth of tables in such case 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 Φ with dimension d. Partition the hyperrectangle $[0,1]^d$ 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 Φ 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 keeping one table per non-empty bucket we are guaranteed not to introduce a local error of more than s. We call this approach AkPU. This solution can be improved by any greedy algorithm for clustering under absolute-norm or Euclidean norm (e.g., k-means).

Theorem 3. *Consider a LIMID of treewidth w and maximum variable cardinality c, and let r be the value computed by AkPU (in fact, with or without dominance pruning) on that LIMID, choosing M at every step in a way that $s \stackrel{def}{=} \lfloor M^{-1/d} \rfloor$ 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 Φ' and Ψ' obtained by bucketing of set-potentials Φ and Ψ, 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 ϕ' and ψ' are in the same buckets as ϕ and ψ, 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 passage we assumed that $\phi(x) \geq \psi(x)$ (otherwise $\gamma(x) - \gamma'(x) \leq 0$, contradicting our initial claim) and used Proposition 2 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 passage we used Proposition 2 to bound expression $\phi(x) + \psi(x)$ in two. Hence, for any γ in Γ there is γ' in Γ' such that $|\gamma(x) - \gamma'(x)| \leq 3m$. Moreover, if Γ'' is a set-potential produced by bucketing of Γ' that for any γ in Γ there is γ'' in Γ'' 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 Γ produced by Y-marginalization of a set-potential $\Phi(\mathcal{X} \cup \{Y\})$, and let Φ' be the output of bucketing Φ. For any $\gamma = \sum_Y \phi$ in Γ there is $\gamma' = \sum_Y \phi'$ in Γ' 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 4. *Given a LIMID of treewidth bounded by a constant w and whose variables have cardinalities bounded by a constant c, and $\epsilon > 0$, AkPU returns a strategy δ_ϵ 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 AkPU 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 α and β are some integer constants. Hence, AkPU 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 AkPU and $s \stackrel{\text{def}}{=} \lfloor M^{-1/d} \rfloor$. By Theorem 3, 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 3 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 Figure 1, $n = 100$ 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, and consideration of the propagated error estimates. This way, M can be adjusted adaptively, demanding much less computational resources than in the proof of Theorem 4, but still guaranteeing a maximum error in the output in fully polynomial time.

6 Experiments

We compared the performance of DkPU and AkPU in a large set of randomly sampled LIMIDs with graph structure as in Fig. 1. While the choice of a fixed structure might seem restrictive, we note that any diagram can be transformed into a diagram like that of Fig. 1 by merging and adding variables [18]. The algorithms were implemented in Python and ran using the Pypy interpreter.[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 $[0, 1]$.

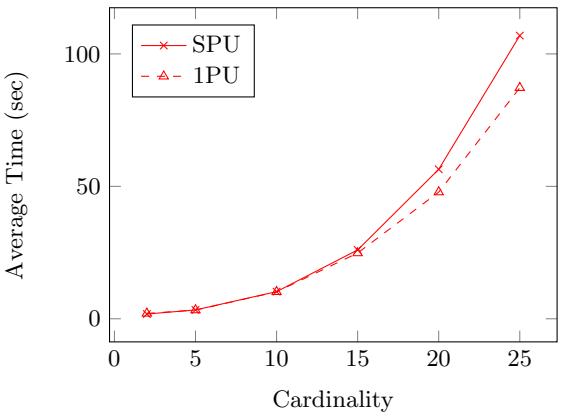

Fig. 2. Comparison between SPU and D1PU

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 in SPU's speed is welcome. Figure 2 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 runs).

Gains in speed with respect to kPU 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 Figures 3 and 4 show that

[1] The code and diagrams are available at `http://github.com/denismaua/kpu`.

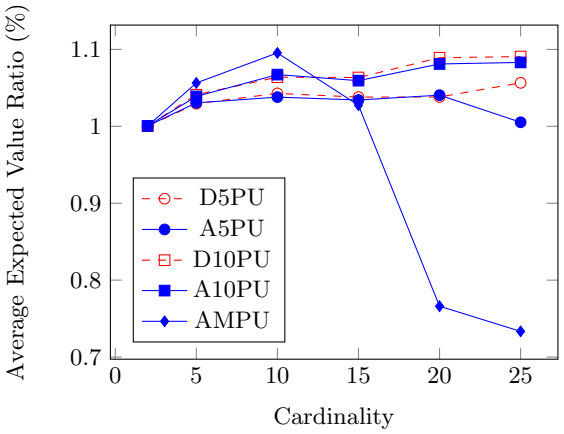

Fig. 3. Average accuracy relative to SPU

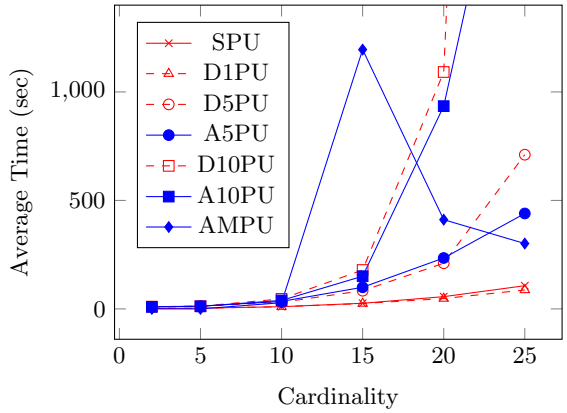

Fig. 4. Average time performance

DkPU can be used up to relatively high k with gains in expected value that reach 10%; also, the move to AkPU does control computing time while still leading to gains in expected value.

Figure 3 shows the average of the ratio between expected value obtained with k-local search schema to expected value obtained with 1-local search. Points higher than one indicate that the corresponding method outperforms 1-local search on average. Dashed curves report the performance of DkPU, whereas the solid curves report the performance of AkPU. We see that the bucketing does not decrease performance significantly, except for the approximate MPU variant, whose accuracy decays considerably with the increase of variable cardinality.

Figure 4 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 than the approximate MPU (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 10-local search methods with and without bucketing 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 hour) was one order of magnitude greater than A10PU (which took about an hour).

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 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 random diagrams of bounded treewidth and varying variable cardinality showed that dominance pruning speeds up computations even for 1-neighborhood local search (i.e., SPU). Hence dominance pruning is *always* useful when one wishes to resort to local search. 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 cardinalities up to 15 or so.

Acknowledgements. The first author is partially supported by the FAPESP grant no. 13/23197-4. The second author is partially supported by CNPq.

References

1. Cooper, G.F.: A method for using belief networks as influence diagrams. In: Fourth Workshop on Uncertainty in Artificial Intelligence (1988)
2. de Campos, C.P., Ji, Q.: Strategy selection in influence diagrams using imprecise probabilities. In: Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI), pp. 121–128 (2008)

3. Dechter, R.: An anytime approximation for optimizing policies under uncertainty. In: Workshop of Decision Theoretic Planning, AIPS (2000)
4. Detwarasiti, A., Shachter, R.D.: Influence diagrams for team decision analysis. Decision Analysis 2(4), 207–228 (2005)
5. Downey, R., Fellows, M.R.: Fixed-parameter tractability and completeness I: Basic theory. SIAM Journal of Computing 24, 873–921 (1995)
6. Downey, R., Fellows, M.R.: Fixed-parameter tractability and completeness II: Completeness for W[1]. Theoretical Computer Science A 141, 109–131 (1995)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
8. Howard, R.A., Matheson, J.E.: Influence diagrams. In: Readings on the Principles and Applications of Decision Analysis, pp. 721–762. Strategic Decisions Group (1984)s
9. Jensen, F.V., Nielsen, T.D.: Bayesian Networks and Decision Graphs, 2nd edn. Information Science and Statistics. Springer (2007)
10. Khaled, A., Yuan, C., Hansen, E.: Solving limited memory influence diagrams using branch-and-bound search. In: Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence, UAI (2013)
11. Kohlas, J.: Information Algebras: Generic Structures for Inference. Springer, New York (2003)
12. Kwisthout, J.H.P., Bodlaender, H.L., van der Gaag, L.C.: The Necessity of Bounded Treewidth for Efficient Inference in Bayesian Networks. In: Proceedings of the 19th European Conference on Artificial Intelligence, pp. 237–242 (2010)
13. Lauritzen, S.L., Nilsson, D.: Representing and solving decision problems with limited information. Management Science 47, 1235–1251 (2001)
14. Liu, Q., Ihler, A.: Belief propagation for structured decision making. In: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI), pp. 523–532 (2012)
15. Mauá, D.D., de Campos, C.P.: Solving decision problems with limited information. In: Advances in Neural Information Processing Systems 24 (NIPS), pp. 603–611 (2011)
16. Mauá, D.D., de Campos, C.P., Zaffalon, M.: On the complexity of solving polytree-shaped limited memory influence diagrams with binary variables. Artificial Intelligence 205, 30–38 (2013)
17. Mauá, D.D., de Campos, C.P., Zaffalon, M.: The complexity of approximately solving influence diagrams. In: Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI), pp. 604–613 (2012)
18. Mauá, D.D., de Campos, C.P., Zaffalon, M.: Solving limited memory influence diagrams. Journal of Artificial Intelligence Research 44, 97–140 (2012)
19. Park, J.D., Darwiche, A.: Complexity results and approximation strategies for MAP explanations. Journal of Artificial Intelligence Research 21, 101–133 (2004)
20. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82(1-2), 273–302 (1996)
21. Szeider, S.: The parameterized complexity of k-flip local search for SAT and MAX SAT. Discrete Optimization 8(1), 139–145 (2011)
22. Tatman, J.A., Shachter, R.D.: Dynamic programming and influence diagrams. IEEE Transactions on Systems, Man and Cybernetics 20(2), 365–379 (1990)