Initialization and Restart in Stochastic Local Search: Computing a Most Probable Explanation in Bayesian Networks

Ole J. Mengshoel, *Member*, *IEEE*, David C. Wilkins, and Dan Roth, *Member*, *IEEE*

Abstract—For hard computational problems, stochastic local search has proven to be a competitive approach to finding optimal or approximately optimal problem solutions. Two key research questions for stochastic local search algorithms are: Which algorithms are effective for initialization? When should the search process be restarted? In the present work, we investigate these research questions in the context of approximate computation of most probable explanations (MPEs) in Bayesian networks (BNs). We introduce a novel approach, based on the Viterbi algorithm, to explanation initialization in BNs. While the Viterbi algorithm works on sequences and trees, our approach works on BNs with arbitrary topologies. We also give a novel formalization of stochastic local search, with focus on initialization and restart, using probability theory and mixture models. Experimentally, we apply our methods to the problem of MPE computation, using a stochastic local search algorithm known as Stochastic Greedy Search. By carefully optimizing both initialization and restart, we reduce the MPE search time for application BNs by several orders of magnitude compared to using uniform at random initialization without restart. On several BNs from applications, the performance of Stochastic Greedy Search is competitive with clique tree clustering, a state-of-the-art exact algorithm used for MPE computation in BNs.

Index Terms—Stochastic local search, Bayesian networks, initialization, restart, finite mixture models.

1 Introduction

Multivariate probability distributions play a central role in a wide range of automated reasoning and state estimation applications. Multivariate probability distributions can be decomposed by means of Bayesian networks [1], factor graphs [2], Tanner graphs, Markov random fields [3], [4], arithmetic circuits [5], or clique trees [6], [7], [8]. If the resulting graph decomposition is relatively sparse, efficient reasoning and learning algorithms exist.

In this paper, we focus on reasoning in the form of stochastic local search in Bayesian networks (BNs). Stochastic local search (SLS) algorithms are among the best known for computationally hard problems including satisfiability (SAT) [9], [10], [11], [12]. SLS algorithms have also been successful in computing the most probable explanation [13], [14], [15], [16], [17], [18] and the maximum a posteriori hypothesis [19] in Bayesian networks. While the details of different SLS algorithms vary [12], by definition they use noise and initialization algorithms in addition to hill-climbing; SLS algorithms typically also rely on restarts.

Our focus in this work is on initialization and restart. Specifically, we investigate the following research questions: How do different SLS initialization algorithms impact performance? How can their effect be analyzed? and What is the impact of the restart parameter? In answering these questions, we consider the Stochastic Greedy Search (SGS) algorithm, an SLS approach for computing MPEs in BNs [14], [15]. The stochastic local search part of SGS is a generalization of the GSAT and WALKSAT family of algorithms [9], [20], [10] to the probabilistic setting of BNs, and SGS is also related to other SLS algorithms for BN computation [21], [13], [19]. Specifically, our contribution in this work is twofold. Our first contribution consists of two novel initialization algorithms. These algorithms are generalizations of the Viterbi approach [22] and use Viterbi to exactly compute MPEs for tree-structured BNs. Algorithms that exploit tree-structured graphs or subgraphs are well known [2], [3], [4]; in this paper, we discuss how such algorithms can be used for SLS initialization in general BNs. Informally, these algorithms construct explanations that are good approximations to MPEs, and these explanations make up starting points for the hill-climbing phase of SGS. In application BNs, we show experimentally that these and other initialization algorithms can substantially improve performance. Our second contribution rests on the belief that the research questions raised above are best answered within a solid mathematical framework. Consequently, we carefully develop a mathematical framework for SLS analysis, based on probability theory and mixture distributions, and fit mixture models to SLS run-length data in experiments. We thus help improve the theoretical foundations of SLS; this is significant because the theoretical foundations of SLS have lagged compared to the impressive experimental performance of these algorithms [23], [24], [17].

The most closely related research to ours has focused on runtime variability, SLS restart, and SLS initialization. An

- *O.J. Mengshoel is with Carnegie Mellon University, NASA-Ames Research Center, Mail Stop 269-3, Bldg. T35-B, Rm. 107, PO Box 1, Moffett Field, CA 94035-0001. E-mail: ole.mengshoel@sv.cmu.edu.*
- *D.C. Wilkins is with Symbolic Systems Program, Stanford University, Bldg 460 Room 127, Stanford, CA 94305. E-mail: dwilkins@stanford.edu.*
- *D. Roth is with the Department of Computer Science, University of Illinois at Urbana-Champaign, 3322 Siebel Center, 201 N. Goodwin Avenue, Urbana, IL 61801. E-mail: danr@cs.uiuc.edu.*

Manuscript received 25 May 2007; revised 28 Dec. 2009; accepted 3 Mar. 2010; published online 7 June 2010.
Recommended for acceptance by J. Pei.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2007-05-0237.
Digital Object Identifier no. 10.1109/TKDE.2010.98.

easy-hard-easy runtime pattern has been observed for the NP-complete satisfiability (SAT) problem [25], and great variability in runtimes has been observed in the hard region of SAT [26]. Even in the easy region, there is great variability in the runtime for an ensemble of problem instances [27]. Based on these observations, the benefit of randomized restarts has been established, both for SLS [28] and for systematic search [29]. Motivated by high runtime variability for hard problem instances, Bayesian models that predict inference runtimes have been learned and then used to dynamically optimize the SLS restart point [30]. In other related research, offline and online computations are combined to dynamically control restarts [31], and restart policies based on beliefs about problem instance hardness are used to make search more efficient [32] . Finally, we note that initialization has turned out to be crucial in making SLS competitive with other approaches to BN computation, especially in application BNs [14], [15], [19], [13].

The rest of this paper is organized as follows: Section 2 contains notation and fundamental concepts. In Section 3, we introduce our SLS approach including our novel initialization algorithms. In Section 4, we analyze our approach using techniques from probability theory and finite mixtures. Experimental results are presented in Section 5. In Section 6, we conclude and present directions for future research. We note that earlier versions of this research have been reported previously [14], [15].

2 Preliminaries

This section introduces notation and known results related to Bayesian networks, the Viterbi approach, and tree-based reparameterization.

2.1 Fundamental Concepts and Notation

Bayesian networks [1], defined as follows, organize random variables in directed acyclic graphs (DAGs):

Definition 1 (Bayesian network). *A Bayesian network is a tuple $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$, where $(\boldsymbol{X}, \boldsymbol{E})$ is a DAG with $n = |\boldsymbol{X}|$ nodes, $m = |\boldsymbol{E}|$ edges, and an associated set of conditional probability distributions $\boldsymbol{P} = \{\Pr(X_1 \mid \Pi_{X_1}), \ldots, \Pr(X_n \mid \Pi_{X_n})\}$. Here, $\Pr(X_i \mid \Pi_{X_i})$ is the conditional probability distribution for $X_i \in \boldsymbol{X}$. Further, let π_{X_i} represent the instantiation of the parents Π_{X_i} of X_i. The independence assumptions encoded in $(\boldsymbol{X}, \boldsymbol{E})$ imply the joint probability distribution*

$$\Pr(\boldsymbol{x}) = \prod_{i=1}^{n} \Pr(x_i \mid \pi_{X_i}), \tag{1}$$

where $\Pr(\boldsymbol{x}) = \Pr(X_1 = x_1, \ldots, X_n = x_n)$.

$\Pr(X_i \mid \Pi_{X_i})$ is also known as a conditional probability table (CPT). The notation Ω_X is used to represent the (here discrete) state space of a BN node X. A BN may be given *evidence* by clamping some nodes to their observed states. An instantiation of the remaining nodes is an explanation, formally defined as follows:

Definition 2 (Explanation). *Consider a BN $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$ with evidence $\boldsymbol{e} = \{x_1, \ldots, x_m\} = \{X_1 = x_1, \ldots, X_m = x_m\}$. An explanation $\boldsymbol{x}$ is defined as $\boldsymbol{x} = \{x_{m+1}, \ldots, x_n\} = \{X_{m+1} = x_{m+1}, \ldots, X_n = x_n\}$. A subexplanation $\boldsymbol{y}$ of $\boldsymbol{x}$ is defined as $\boldsymbol{y} \subseteq \boldsymbol{x}$.*

To simplify the exposition, one may regard $\boldsymbol{z} = \boldsymbol{x} \cup e$, and consider $\Pr(\boldsymbol{z}) = \Pr(\boldsymbol{x}, \boldsymbol{e}) = \Pr(\boldsymbol{x} \mid \boldsymbol{e}) \Pr(\boldsymbol{e})$ instead of the closely related $\Pr(\boldsymbol{x} \mid e)$. The BN β is typically left implicit when discussing an explanation $\boldsymbol{x}$ for β.

Given a BN with evidence or no evidence, various forms of BN inference can be performed [1], [6], [7], [33], [8], [19]. This paper focuses on computing the most probable explanation, also known as belief revision [1].

Definition 3 (Most probable explanation (MPE)). *Computing an MPE in a BN is the problem of finding an explanation $\boldsymbol{x}^*$ such that $\Pr(\boldsymbol{x}^*) \geq \Pr(\boldsymbol{y})$, where $\boldsymbol{y}$ is any other explanation in the BN. The set of the k most probable explanations is defined as $\boldsymbol{X}^* = \{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_k^*\}$, where $\Pr(\boldsymbol{x}^*) = \Pr(\boldsymbol{x}_1^*) = \cdots = \Pr(\boldsymbol{x}_k^*)$.*

In other words, no other explanation has higher probability than $\boldsymbol{x}_i^*$ for $1 \leq i \leq k$. Several explanations with the same probability can exist, and we therefore say "an" MPE rather than "the" MPE.

It can be shown by reduction from SAT that MPE computation is NP-hard [34]. Approximating an MPE to within a constant ratio-bound has also been proven to be NP-hard [35]. Since inference in BNs is computationally hard and the MPE problem is important in applications, it is important to study inexact approaches, including SLS algorithms, where estimates of $\boldsymbol{x}^* \in \boldsymbol{X}^*$ are computed.

Definition 4 (MPE (lower bound) estimate). *Let $\boldsymbol{x}^*$ be an MPE. A best-effort estimate of $\boldsymbol{x}^*$ is denoted $\hat{\boldsymbol{x}}^*$; if $\Pr(\hat{\boldsymbol{x}}^*) \leq \Pr(\boldsymbol{x}^*)$ then $\hat{\boldsymbol{x}}^*$ is a lower bound estimate.*

SLS algorithms typically compute lower bound MPE estimates $\hat{\boldsymbol{x}}^*$. In Section 3.1, we discuss one SLS algorithm, Stochastic Greedy Search, in more detail.

2.2 Existing Dynamic Programming Algorithms

We now discuss forward and backward dynamic programming for chains and trees; we denote these algorithms TREEFDP and TREEBDP, respectively. We present this well-known approach due to Viterbi [36], [22], for BNs that are chains in some detail. The case of trees is a straightforward extension since different paths down a tree are independent and can be treated independently using essentially the same algorithm. We introduce the following terminology for BNs with such tree topologies.

Definition 5 (Backward tree, forward tree). *Consider a BN $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$. If the underlying graph $(\boldsymbol{X}, \boldsymbol{E})$ of β is a tree where all nonleaf nodes have one or more children, then β is a backward tree. If the underlying graph $(\boldsymbol{X}, \boldsymbol{E})$ of β is a tree where all nonroot nodes have one or more parents, then β is a forward tree.*

$Y \rightarrow X \leftarrow Z$ is an example forward tree and $Y \leftarrow X \rightarrow Z$ is an example backward tree; see Fig. 2 for other backward tree examples.

To simplify exposition, we now assume BN nodes with $S = 2$ states, say $\{0, 1\}$. The approach clearly generalizes to $S > 2$. We consider a BN that is a chain $X_1 \rightarrow X_2 \rightarrow \cdots \rightarrow$

X_T of length T. The CPTs are denoted by $\Pr(X_1 = x_1)$, $\Pr(X_2 = x_2 \mid X_1 = x_1)$, ..., $\Pr(X_t = x_t \mid X_{t-1} = x_{t-1})$, where $x_i \in \{0, 1\}$. The key observation that TREEFDP is based on is the following. In order to compute the probability of the most probable explanation of a subchain of length $t \leq T$, it is sufficient to know $S = 2$ numbers: 1) the probability of the most probable explanation until node number $t-1$, assuming that the $(t-1)$th node is set to 0; and 2) the probability of the most probable explanation until node number $t-1$, assuming that the $(t-1)$th node is set to 1.

More specifically, let $x_t \in \{0, 1\}$ be the assignment of the tth node. In order to compute an MPE, we operate on two arrays, the D-array containing probabilities (such as $\Pr(\boldsymbol{x}^*)$) and the A-array containing states (such as $\boldsymbol{x}^*$). Let $D_t(x_t)$ be the maximal probability of a sequence of assignments to nodes $X_1, \ldots, X_t$, with $X_t = x_t$. Let $A_{t-1}(x_t)$ be the assignment to the node X_{t-1} in the most probable subexplanation that assigns $X_t = x_t$.

For initialization in the TREEFDP algorithm, let $D_1(x_1) = \Pr(x_1)$ for $x_1 \in \{0, 1\}$. These D_1 values are simply priors. Also set $A_0(x_0) = \{\}$. The recursive step is for the D-array

$$D_t(x_t) = \max_{x_{t-1} \in \{0,1\}} \{D_{t-1}(x_{t-1}) \Pr(x_t \mid x_{t-1})\},$$

while for the A-array, we get the recursive step

$$A_{t-1}(x_{t-1}) = \arg \max_{x_{t-1} \in \{0,1\}} \{D_{t-1}(x_{t-1}) \Pr(x_t \mid x_{t-1})\}.$$

The final assignments in the iterative loop are to $D_T(X_T = 0)$ and $D_T(X_T = 1)$ for probabilities, and to $A_{T-1}(X_T = 0)$ and $A_{T-1}(X_T = 1)$ for states.

Computing $D_T = \max_{x_T \in \{0,1\}} \{D_T(X_T = x_T)\}$ is one of the last steps of the algorithm. Here, we choose the best of the two probabilities $D_T(X_T = 0)$ and $D_T(X_T = 1)$. For the states, we similarly compute the last assignment $A_T = \arg\max_{x_T \in \{0,1\}} \{D(X_T = x_T)\}$. Note that this is done after computing $A_{T-1}(x_T)$. Given the information in the A-array, one just needs to perform a backtracking step in order to compute the MPE $\boldsymbol{x}^*$ for the chain [22, p. 264, (35)].

The TREEBDP algorithm is similar to TREEFDP. Again, there are two arrays, an array E containing probabilities, and an array B containing states. The E-array corresponds to the D-array, while the B-array corresponds to the A-array. To save space, we refer to the literature [36], [22] for further details.

The following result, adapted from Rabiner [22], summarizes the performance of TREEBDP and TREEFDP.

Theorem 1. *In a BN that is a backward tree, TREEBDP computes an MPE $\boldsymbol{x}^*$. In a BN that is a forward tree, TREEFDP computes an MPE $\boldsymbol{x}^*$.*

Viterbi, generalized to arbitrary tree-structured graphical models, is called max-product belief propagation [2]. A tree-based reparameterization framework has been developed [3], [4], which includes the belief propagation [1] and sum-product algorithms [2] as special cases. Within this reparameterization framework, there are approximation algorithms for MPEs [3] and marginals [4]. Our novel approaches to initialization, discussed in Section 3.2.2, are based on TREEBDP and TREEFDP and are closely related to the tree-based reparameterization algorithm for MPE computation.

```
SGS(β, ℓ, 𝕀, 𝕊, MAX-FLIPS, MAX-TRIES)
  Input:  β          Bayesian network (BN)
          ℓ          lower bound: Pr(x̂*) ≥ ℓ
          𝕀          initialization portfolio
          𝕊          local search portfolio
          MAX-FLIPS  number of flips per try
          MAX-TRIES  number of tries
  Output: (b, x̂*)    b ∈ {true,false};
                     x̂* is estimated optimum
begin
    x̂* ← INITIALIZE(𝕀, β) {initial MPE estimate x̂*}
    if (Pr(x̂*) ≥ ℓ) then return (true, x̂*)
    i ← 1
    while (i ≤ MAX-TRIES)
        j ← 0 {the 0th operation is this initialization}
        x ← INITIALIZE(𝕀, β) {initialize x}
        if (Pr(x̂*) ≥ ℓ) then return (true, x̂*)
        j ← 1
        while (j ≤ MAX-FLIPS)
            x ← SEARCH(x, 𝕊, β) {update x}
            if (Pr(x) > Pr(x̂*)) then x̂* ← x
            if (Pr(x̂*) ≥ ℓ) then return (true, x̂*)
            j ← j + 1
        endwhile
        i ← i + 1
    endwhile
    return (false, x̂*)
end
```

Fig. 1. The stochastic local search algorithm, SGS, for computing MPEs. SGS operates in two main phases: an initialization phase and a local search phase. INITIALIZE applies initialization algorithms from $\mathbb{I}$, while SEARCH applies search algorithms from $\mathbb{S}$. SGS terminates if a high-probability explanation is found or if the number of tries exceeds MAX-TRIES.

3 Stochastic Local Search

In this section, we present our stochastic local search algorithm, SGS, in Section 3.1. Our initialization algorithms, the forward dynamic programming (GRAPHFDP) and backward dynamic programming (GRAPHBDP) algorithms, are discussed in Section 3.2.

3.1 Stochastic Greedy Search

Fig. 1 presents our SGS algorithm, discussed in more detail elsewhere [15], [17], [18]. The structure of SGS is similar to that of the seminal WALKSAT family of algorithms [20], [10]. In SGS, as in WALKSAT, the MAX-FLIPS restart parameter controls the number of flips made before SGS is restarted. A *try* starts with initialization and ends after at most MAX-FLIP flips. After MAX-FLIPS flips, if there is a new try, a new explanation $\boldsymbol{x}$ from which search restarts are randomly generated; this is also similar to WALKSAT. If exactly one initialization is performed at the beginning of each try, a total of at most MAX-FLIPS + 1 operations are performed per try. Two termination criteria are displayed in Fig. 1; the lower bound $\ell \leq \Pr(\boldsymbol{x}^*)$ and the MAX-TRIES parameter. The latter is easy to understand, for the former SGS terminates when the probability of the MPE estimate $\hat{\boldsymbol{x}}^*$ exceed the lower bound ℓ, or $\Pr(\hat{\boldsymbol{x}}^*) \geq \ell$. While we assume that ℓ is an input parameter, it can also be estimated during search. Other termination criteria can easily be introduced.

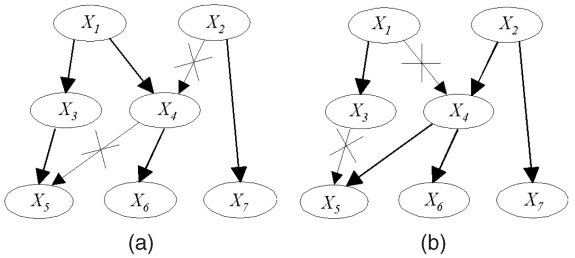

Fig. 2. Backward dynamic programming initialization in a nontree Bayesian network. (a) The BN is decomposed, by GRAPHBDP, into two backward trees $\mathbf{T_1} = \{X_1, X_3, X_4, X_5, X_6\}$ and $\mathbf{T_2} = \{X_2, X_7\}$. (b) The BN is decomposed, again by GRAPHBDP, into two different backward trees $\mathbf{T_1} = \{X_1, X_3\}$ and $\mathbf{T_2} = \{X_2, X_4, X_5, X_6, X_7\}$.

There are some important differences between SGS and many other WALKSAT-style algorithms. First, SGS searches for an MPE in a BN, not a satisfying assignment in a CNF logic formula. Second, SGS combines greedy and noisy search algorithms in a portfolio $\mathbb{S}$ with stochastic initialization algorithms in a portfolio $\mathbb{I}$, while WALKSAT does not use portfolios. Formally, we use the following definition (see [18]):

Definition 6 (Stochastic portfolio). *Let $q \geq 1$ and let $\Phi = \{\phi_1, \ldots, \phi_q\}$ be a set of algorithms. A stochastic portfolio over Φ is a set of q tuples $\Lambda = \{\nu_1, \ldots, \nu_q\} = \{(\phi_1, p_1), \ldots, (\phi_q, p_q)\}$ where $0 \leq p_i \leq 1$, $\sum_{i=1}^{q} p_i = 1$, and (ϕ_i, p_i) means that the ith algorithm ϕ_i, where $1 \leq i \leq q$, is picked (and executed) with selection probability p_i when an algorithm is selected from Λ.*

Both $\mathbb{I}$ and $\mathbb{S}$ are portfolios according to this definition.

3.2 Initialization Algorithms

Initialization algorithms play an important role in SGS and other SLS algorithms [14], [15], [16]. We present below several initialization algorithms for stochastic generation of initial explanations. SGS uses a portfolio $\mathbb{I}$ of initialization algorithm (or operators), and initialization of an explanation takes place when INITIALIZE($\mathbb{I}$, β) is invoked. The algorithms discussed in this section, as well as other initialization algorithms, can easily be incorporated in the initialization portfolio $\mathbb{I}$ of SGS. Evidence $\boldsymbol{e}$ in a BN is not changed by any of these initialization algorithms, however, in order to simplify the discussion, we often do not make this explicit in the following.

3.2.1 Related SLS Initialization Algorithms

Uniform initialization (UN), the basis for our SGS/UN experiments in Section 5, assigns initial states independently and uniformly at random to an explanation $\boldsymbol{x}$. More formally, suppose that we have a Bayesian network with nodes $\boldsymbol{X}$. The uniform initialization algorithm goes through all nodes $X \in \boldsymbol{X}$. If X is a nonevidence node, each state $x \in \Omega_X$ has a probability of $1/|\Omega_X|$ of being chosen to be part of the initial explanation $\boldsymbol{x}$. An early approach to MPE computation by means of stochastic methods used uniform initialization and investigated three randomized search techniques: iterative local search, simulated annealing, and genetic search [21]. In general, SAT solvers have also employed initialization uniformly at random, and innovations have largely been made in the area of search heuristics.

In the late 1990s, the benefit of more advanced SLS initialization algorithms when computing an MPE became clear [13], [14], [15].

Forward simulation (FS), the basis for our SGS/FS experiments in Section 5, is a well-known BN simulation algorithm that operates as follows [37]: Suppose we have a Bayesian network where $\boldsymbol{V}$ represents the root nodes and $\boldsymbol{C}$ represents the nonroot nodes. Using FS, a root node $V \in \boldsymbol{V}$ is initialized, i.e., its states chosen for inclusion in the initial explanation $\boldsymbol{x}$, independently at random according to the prior distribution $\Pr(V)$. A nonroot node $C \in \boldsymbol{C}$ is initialized randomly according to its conditional distribution $\Pr(C \mid \Pi_C)$, and only after all of its parent nodes Π_C have been initialized. Clearly, both UN and FS are $O(n)$.

Kask and Dechter empirically found strong MPE performance using greedy search combined with stochastic simulation [1] after performing initialization using the minibucket approximation algorithm [13]. This algorithm [38] approximates bucket elimination and is useful for problem instances with large induced width (or treewidth), since bucket elimination has exponential space and time complexity in induced width [39].

Developing an SLS approach for the MAP problem, which generalizes the MPE problem for BNs, Park and Darwiche investigated two search algorithms (hill-climbing and taboo search) as well as four initialization algorithms [19]. Since the MAP problem is strictly harder than the MPE problem [19], they in fact use MPE computation as an initialization algorithm.

3.2.2 Novel Dynamic Programming Algorithms

We now turn to GRAPHFDP and GRAPHBDP. Unlike TREEBDP and TREEFDP, GRAPHFDP and GRAPHBDP can handle arbitrary BNs and not only trees and chains. Both these algorithms split a BN β into trees, initialize each tree independently using Viterbi (see Section 2.2), and then collect the subexplanations for all trees to give an explanation for the entire BN β. More formally, let $\boldsymbol{X}$ be the nodes in a BN. These nodes are partitioned into k partitions

$$\boldsymbol{X} = \boldsymbol{T}_1 \cup \ldots \cup \boldsymbol{T}_k, \tag{2}$$

where $\boldsymbol{T}_i \cap \boldsymbol{T}_j = \emptyset$ for $i \neq j$. All partitions $\boldsymbol{T}_i$, where $1 \leq i \leq k$, are either forward trees or backward trees.

Without loss of generality, we now assume that GRAPHBDP is used and that all $\boldsymbol{T}_i$, where $1 \leq i \leq k$, therefore are backward trees. Consider the ith backward tree $\boldsymbol{T}_i = \{X_{i_1}, \ldots, X_{i_N}\}$, where $X_{i_j} \in \boldsymbol{X}$. For $\boldsymbol{T}_i$, TREEBDP computes the subexplanation

$$\boldsymbol{x}_i^* = \{X_{i_1} = x_{i_1}, \ldots, X_{i_N} = x_{i_N}\}, \tag{3}$$

which is an MPE for $\boldsymbol{T}_i$ according to Theorem 1. An MPE estimate $\hat{\boldsymbol{x}}^*$ for the complete BN β can now be generated by GRAPHBDP simply by collecting subexplanations for all k backward trees:

$$\hat{\boldsymbol{x}}^* = \boldsymbol{x}_1^* \cup \ldots \cup \boldsymbol{x}_k^*. \tag{4}$$

The GRAPHBDP algorithm is presented in Fig. 3.

In constructing the trees $\{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_k\}$, some way to introduce randomness is clearly needed in GRAPHBDP. That is, we do not want every explanation generated using INITIALIZE to be the same. Randomness is introduced by constructing depth-first search trees where root nodes, and

```
GRAPHBDP(β,e)
 Input:  β  Bayesian network (BN)
         e  evidence
 Output: x  explanation
begin
    V ← root nodes in β
    F ← ∅ {initialize the forest of backward trees F}
    while V ≠ ∅ {treat all root nodes}
        V ← pick random root node from V
        V ← V − {V}
        {construct backward tree T with root V:}
        T ← STOCHASTICDFS(β, V, e) {see text}
        F ← F ∪ {T} {update forest F of trees}
    endwhile
    x ← ∅ {initialize the explanation x}
    V ← root nodes in β
    while V ≠ ∅ {treat all root nodes}
        V ← pick node from V
        V ← V − {V}
        T ← pick, from F, the tree with root V
        y ← TREEBDP(β, T, e)
        x ← x ∪ {y} {update explanation}
    endwhile
    return x
end
```

Fig. 3. The dynamic programming algorithm that iteratively creates backward trees for a BN and then executes the Viterbi algorithm separately on each of these trees.

then the children of any node, are picked for processing uniformly at random. We call this novel approach stochastic depth-first search, STOCHASTICDFS, and summarize it as follows:

Let $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$ be a BN, $V \in \boldsymbol{X}$ a root node, and $\boldsymbol{e}$ evidence. The algorithm STOCHASTICDFS $(\beta, V, \boldsymbol{e})$ performs a depth-first search where all nonvisited children of V are recursively visited uniformly at random. STOCHASTICDFS outputs a backward tree $\boldsymbol{T}_i$ rooted in V and marks all nodes in $\boldsymbol{T}_i$ as visited in β. In $\boldsymbol{T}_i$, all nodes reachable from V along a directed path in β are included, except those nodes in β that are part of backward trees $\{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_{i-1}\}$ already created by STOCHASTICDFS and thus already marked as visited.

In its first **while**-loop, GRAPHBDP decomposes a BN β into trees, starting from the root nodes, resulting in a forest of trees. Each tree $\boldsymbol{T}$ in the induced forest of trees is then, in the second **while**-loop, input to TREEBDP, and an MPE for $\boldsymbol{T}$ is thus computed. Given the combined operation of STOCHASTICDFS and GRAPHBDP, we can show the following.

Theorem 2. *Let β be a BN and $\boldsymbol{e} = \{X_1 = x_1, \ldots, X_m = x_m\}$ the evidence. The GRAPHBDP $(\beta, \boldsymbol{e})$ algorithm creates a forest of trees $\{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_k\}$ in which each node $X \in \boldsymbol{X}$ of $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$ participate in exactly one tree $\boldsymbol{T}_i$, where $1 \leq i \leq k$.*

Proof. We first show that any node must be a member of at least one tree. In $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$, where $\boldsymbol{V} \subseteq X$ are root nodes, suppose for the purpose of contradiction that $X \in \boldsymbol{X}$ is not in any tree. Obviously, X is either a BN root node, $X \in \boldsymbol{V}$, or X is a BN nonroot node, $X \in \boldsymbol{X} - \boldsymbol{V}$. Case 1: If $X \in \boldsymbol{V}$, it is the root of exactly one tree by construction of GRAPHBDP and there is a contradiction. Case 2: If X is a nonroot node, $X \in \boldsymbol{X} - \boldsymbol{V}$, it is reachable by STOCHASTICDFS from one or more root nodes, say $\{V_1, \ldots, V_m\} \subseteq \boldsymbol{V}$, through its parent nodes Π_X. Some $V_i \in \{V_1, \ldots, V_m\}$ must have been the first to have been picked in the first **while**-loop of GRAPHBDP. At that point, by construction of STOCHASTICDFS, X would eventually have been included in V_i's tree, thus giving a contradiction and proving that any node must be a member of at least one tree. Now we show that $X \in \boldsymbol{X}$ cannot be member of two different trees. Suppose, for the purpose of contradiction, that $X \in \boldsymbol{T}_i$ and $X \in \boldsymbol{T}_j$ with respective root nodes V_i and V_j for $i \neq j$. The only nontrivial case is $X \neq V_i$ and $X \neq V_j$. Without loss of generality, we assume that V_i was picked before V_j from $\boldsymbol{V}$ in the first **while**-loop of GRAPHBDP, so $i < j$. When V_i was picked, and since $X \in \boldsymbol{T}_i$, there must be a path from V_i to X and thus X is marked visited upon inclusion in $\boldsymbol{T}_i$. By construction of STOCHASTICDFS, X cannot be included in another tree $\boldsymbol{T}_j$ for $j > i$, giving a contradiction. An evidence node X_i in $\boldsymbol{e}$ is treated exactly like any other node except that it's state x_i is not changed. We have shown that any node must be a member of at least one tree and cannot be member of two trees; three or more trees follows easily in a similar manner, proving the claim. □

The second **while**-loop in GRAPHBDP invokes TREEBDP for each backward tree $\boldsymbol{T}_i$. For each $\boldsymbol{T}_i$, TREEBDP then sets up the dynamic programming arrays, starting from the leaf nodes. The arrays are one numerical array E and one state array B as discussed in Section 2.2. Then, TREEBDP constructs a subexplanation $\boldsymbol{x}_i^*$—as shown in (3)—by forward propagation and back-tracking over these DP arrays. Finally, all subexplanations are collected to form $\hat{\boldsymbol{x}}^*$; see (4).

Example 1. Fig. 2 illustrates, for a small BN, how GRAPHBDP may decompose a BN into two different backward trees.

Example 1 illustrates the following general result, which follows immediately from Theorem 2 and our discussion above.

Corollary 3. *Consider a BN $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$ with $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ and evidence $\boldsymbol{e} = \{X_1 = x_1, \ldots, X_m = x_m\}$ where $m < n$. GRAPHBDP$(\beta, \boldsymbol{e})$ computes an explanation $\boldsymbol{y}$ over all nonevidence nodes $\boldsymbol{Y} = \{X_{m+1}, \ldots, X_n\} \subset \boldsymbol{X}$.*

GRAPHFDP works similar to GRAPHBDP, but starts its stochastic decomposition of the BN into a forest of forward trees from the leaf nodes, constructs DP arrays from the root nodes, and then propagates and backtracks. Both GRAPHFDP and GRAPHBDP have complexity $O(n)$, since each node $X \in \boldsymbol{X}$ is processed at most once in both algorithms.

Both GRAPHBDP and GRAPHFDP are heuristics and have limitations. At the same time, this way of generalizing beyond the cases of chains and trees is fast and produces, as it turns out, good explanations for certain tree-like BNs. Since trees are generated in a randomized fashion, different subexplanations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ which are MPEs for the individual trees $\{\boldsymbol{T}_1, \ldots, \boldsymbol{T}_k\}$ are constructed and aggregated to form different candidate MPEs. For trees, we will get an MPE $\boldsymbol{x}^*$; generally an explanation constructed in this manner

is an MPE estimate $\hat{\boldsymbol{x}}^*$. Experimentation is needed to evaluate their quality, and we return to this in Section 5.

4 Theoretical Framework for SLS

SLS algorithms often have highly variable runtimes depending on the initial explanation and may also be restarted at different points of their execution. In this section, we carefully analyze SGS, in particular, with regard to initialization and restart.

4.1 Fundamentals of SLS Search

A number of nonnegative random variables can be introduced to characterize the (pseudo)random behavior of SLS algorithms including SGS. Let, for a try, the number of initialization operations applied be a random variable X and the number of search operations (flips, either greedy or noisy) applied be a random variable Y. The total number of operations applied in a try is then a random variable $Z = X + Y$.

While SGS contains an initialization portfolio $\mathbb{I}$, we investigate general initialization portfolios in a related article [17] and focus here on homogenous initialization portfolios. These portfolios, containing a single initialization algorithms, have the form $\mathbb{I} = \{(a_1, 0), \ldots, (a_j, 1), \ldots, (a_\xi, 0)\}$, which can be abbreviated $\mathbb{I} = \{(a_j, 1)\} = \{(a, 1)\}$ or SGS/a. In addition, we want to make the value of the restart parameter MAX-FLIPS $= m$ explicit. To reflect these two points, we say $Z(a, m)$ and $Y(a, m)$ rather than just Z and Y, respectively.[1] We now obtain the expected number of operations in a try

$$E(Z(a,m)) = E(X) + E(Y(a,m)) = 1 + E(Y(a,m)), \quad (5)$$

where the last equality holds for SLS algorithms that perform exactly one initialization per try, as will be assumed in the following. The notation $m = \infty$ means that there is no restart.

SGS stops a try when an explanation $\hat{\boldsymbol{x}}^*$ such that $\Pr(\hat{\boldsymbol{x}}^*) \geq \ell$, which we abbreviate $\hat{\boldsymbol{x}}^*_\ell$, is found or when the cutoff MAX-FLIPS is reached. In the former case, we say that the try is successful, and define success probability $p_s(a, m)$ accordingly. For the latter case, we define failure probability $p_f(a, m)$.

Definition 7 (Success, failure of try). *Let MAX-FLIPS $= m$. The success probability of an SLS try is*

$$p_s(a,m) := \sum_{i=0}^{m} \Pr(Y(a,\infty) = i). \quad (6)$$

The failure probability is $p_f(a, m) := 1 - p_s(a, m)$.

An MPE estimate $\hat{\boldsymbol{x}}^*_\ell$ might be computed without much search, if initialization is strong relative to the problem instance. Therefore, summation starts with $i = 0$ in (6) to capture the probability of finding an $\hat{\boldsymbol{x}}^*_\ell$ as a result of initialization. When there is a good fit between a problem instance and an initialization algorithm, $\Pr(Y(a, \infty) = 0)$ can in fact be substantial, as we will see in experiments in Section 5.

1. We could also have added SGS's other input parameters to Z and Y, however, this would have made for a too tedious notation for the purpose of this paper, where we focus on initialization and restart.

The expected number of SGS operations executed, $E(Z(a, m))$ is characterized by the following result. The assumption $p_s(a, m) > 0$ made below is reasonable since if $p_s(a, m) = 0$ then using MAX-FLIPS $= m$ is futile.

Theorem 4 (Expected number of operations). *Suppose that $p_s(a, m) > 0$ and let MAX-FLIPS $= m$. The expected number of SLS operations executed during a run, $E(Z(a, m))$, is given by*

$$E(Z(a,m)) = \frac{m p_f(a,m) + \sum_{i=0}^{m} i \Pr(Y(a,\infty) = i) + 1}{p_s(a,m)}. \quad (7)$$

Proof. We introduce an indicator random variable R, and let $R = 1$ mean SLS success in the first try while $R = 0$ means SLS failure in the first try. Using conditional expectation gives

$$\begin{aligned} E(Z(a,m)) = {} & E(Z(a,m) \mid R = 0) \Pr(R = 0) \\ & + E(Z(a,m) \mid R = 1) \Pr(R = 1). \end{aligned} \quad (8)$$

First, we consider the case where SLS terminates in the first try and obtain

$$E(Z(a,m) \mid R = 1) = 1 + \sum_{i=0}^{m} \frac{i \Pr(Y(a,\infty) = i)}{p_s(a,m)}, \quad (9)$$

where the first term is due to initialization and the second term is due to flips and is normalized using $p_s(a, m)$ since we assume that SLS terminates in this try. Second, we consider the case where SLS does not terminate in the first try and obtain $E(Z(a, m) \mid R = 0) = E(Z(a, m) + m + 1)$, which by linearity is

$$E(Z(a,m) \mid R = 0) = E(Z(a,m)) + m + 1. \quad (10)$$

Substituting (9), (10), $\Pr(R = 0) = p_f(a, m)$, and $\Pr(R = 1) = p_s(a, m)$ into (8) and then solving for $E(Z(a, m))$ gives the desired result for a run (7). □

Results similar to Theorem 3 have been obtained and discussed previously [28], [40]. Theorem 3 is novel in that we account for all operations including initialization, not only flips $Y(a, m)$ as earlier [28], [40].[2] This is important when initialization makes up a significant part of total computation time, which is the case when small values of MAX-FLIPS are optimal or close to optimal. In addition, previous analytical results do not contain an explicit proof as stated above [28], [40].

The distributions of $Z(a, m)$ and $Y(a, m)$ are also known as run length distributions (RLDs) [12].[3] A related random variable, which we will denote $C(a, m)$, measures wall-clock time in, say, milliseconds, and is known as the runtime distribution (RTD). Given the additional variability introduced in RTDs, due to empirical measurements that depend on the software and hardware used in addition to the random number of SLS operations, we mainly use RLDs in this paper.

The performance of SLS algorithms varies dramatically as the value of the restart parameter MAX-FLIPS varies.

2. Notation used previously is slightly different from ours. Instead of $E(Y(m))$, Parkes and Walser use $E_{\nu,m}$, where ν represents a problem instance and MAX-FLIPS $= m$ [28]. Schuurmans and Southey say $E(T)$ with restart after t flips instead of $E(Y(m))$ [40].

3. A more descriptive terminology for $Y(m)$ would perhaps be "flip length distribution," but we will use the more common "run length distribution" in this paper.

How can MAX-FLIPS, then, be set such that SLS performance is optimized? In answering this question, one needs to consider what, exactly, to optimize. It is clearly reasonable to minimize the expected number of SLS operations, and we consequently introduce the following definition.

Definition 8 (Expectation-optimal MAX-FLIPS). *Let MAX-FLIPS $= m$ and let $Z(a, m)$ be the random number of SLS operations. The expectation-optimal value m_o^* for MAX-FLIPS is defined as*

$$m_o^*(a) = \arg\min_{m \in \mathbb{N}}(E(Z(a, m))), \tag{11}$$

with $\mu_o^*(a) = E(Z(a, m_o^*))$, often abbreviated m_o^* and μ_o^*, respectively.

Occasionally, we minimize the expected number of SLS flips $E(Y(a, m))$ instead of minimizing $E(Z(a, m))$ as above, and define $m_f^*(a)$ in a similar manner to $m_o^*(a)$ and also define $\mu_f^*(a) = E(Y(a, m_f^*))$.

4.2 Finite Mixture Models of SLS

We now discuss finite mixture models, which have at least two benefits in the SLS setting. First, finite mixture models turn out to be a reasonable model of the multimodal nature of SLS run length distribution in many instances [26]. Second, finite mixtures allow us to improve the understanding of the role of restarts in SLS.

4.2.1 General Case of Finite Mixture Models

In order to make further progress, and supported by previous research [26] as well as experiments in Section 5, we now introduce the assumption that an RLD maybe characterized as a finite mixture of κ components.

Definition 9 (SLS finite mixture model). *Let MAX-FLIPS $= m$, assume a homogenous initialization portfolio $\{(a, 1)\}$, and let F_i (for $1 \leq i \leq \kappa$) be a cumulative distribution function. An SLS finite mixture model of the RLD $Z(a, m)$ is defined as*

$$\begin{aligned} F(z; a, m) = {} & \pi_1(a, m)F_1(z; a, m) \\ & + \cdots + \pi_\kappa(a, m)F_\kappa(z; a, m), \end{aligned} \tag{12}$$

where $\sum_{i=1}^{\kappa} \pi_i(a, m) = 1$.

Without loss of generality, we assume that the distributions $F_1, \ldots, F_\kappa$ are ordered according to their respective means: $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_\kappa$. Further, when no restart is used we may say $\pi_i(a)F_i(z; a)$ and $Z(a)$, and when the initialization algorithm a is clear from the context or assumed fixed we may say $\pi_i(m)F_i(z; m)$ and $Z(m)$.

An interesting special case of (12) is to model an SLS run length distribution as a two-component finite mixture distribution

$$F(z; a, m) = \pi_1(a, m)F_1(z; a, m) + \pi_2(a, m)F_2(z; a, m). \tag{13}$$

Now, for the case $\ell = \Pr(\boldsymbol{x}^*)$, π_1 and F_1 represent initializations that are close to the MPEs $\boldsymbol{X}^*$, while π_2 and F_2 represent initializations farther away from $\boldsymbol{X}^*$. The characteristics of π_1, F_1, π_2, and F_2 will depend on the RLD at hand, which again depends on the BN, the SLS initialization algorithm, and the other SLS parameters. The idea is that a strong initialization algorithm yields a larger π_1 and an F_1 that is more skewed toward shorter run lengths compared to a weak initialization algorithm.

The formalization above, using finite mixtures, improves our understanding of the following well-known strategy: Initialize and then run the SLS algorithm for a "small" number of steps. If the initial explanation $\boldsymbol{x}$ turned out to be "close" to an optimum, then the SLS algorithm exhibits the $\pi_1 F_1(z)$ case. If the initial explanation $\boldsymbol{x}$ turned out to be "far away" from an optimum, then we have the $\pi_2 F_2(z)$ case. In (13), the term $\pi_1 F_1(z)$ will in general be of greatest interest to us since it represents successful initializations. Intuitively, the idea is to set MAX-FLIPS such that the SLS algorithm uses, in (13), the F_1 distribution repeatedly rather than "wait around" till the F_2 distribution takes effect.

4.2.2 Special Cases of Finite Mixture Models

The specific mixture models we discuss below are for continuous random variables, while in Section 4.1, we considered discrete random variables. Consequently, we note that $E(Z(m))$ (see Theorem 3) can be approximated, using continuous random variables, as

$$E(Z(m)) \approx \frac{m(1 - F(m)) + \int_0^m z f(z) dz + 1}{F(m)}, \tag{14}$$

where $F(z)$ is a cumulative distribution function and $f(z)$ the corresponding probability density function.

We now discuss finite mixtures of exponential distributions, which is of interest for several reasons. First, RLDs that are close to exponential have been observed in experiments [23], [26], [12]. For example, an empirical RLD for a hard SAT instance containing 100 variables and 430 clauses was generated using WALKSAT and found to be well approximated by an exponential distribution with $\hat{\mu} = 61,081.5$ [12]. Second, the exponential distribution, and its discrete counterpart the geometric distribution, are memoryless. Restarting SLS algorithms whose RLDs almost follow these distributions is therefore of little use [40], and they are thus interesting as bounding cases for restart. Third, it is relatively easy to analyze the exponential distribution.

The exponential distribution has only one parameter and its mode is at zero. The probability of finding $\boldsymbol{x}_\ell^*$ within zero or a few flips is, on the other hand, often extremely small (see Figs. 4 and 6 for experimental evidence). For such situations, the normal (or Gaussian) and log-normal distributions, and their finite mixtures, maybe more suitable and are used extensively in Section 5, see Tables 1 and 2. A random variable is log-normal if its logarithm is normally distributed. The normal and log-normal distributions both have two parameters, controlling location and spread, making them well suited to handling the type of right-shifting needed to model the low-probability left tails that can be empirically observed for SLS algorithms including SGS.

5 Experiments with SLS

We now consider experimental results for SGS. Section 5.1 discusses our experimental approach. In Section 5.2, we characterize the behavior of different SLS initialization algorithms, including our novel Viterbi-based approach, using finite mixture models and other techniques. In Sections 5.3 and 5.4, we empirically investigate the effect of restarts, and, in particular, vary and optimize the MAX-FLIPS parameter in the context of different initialization

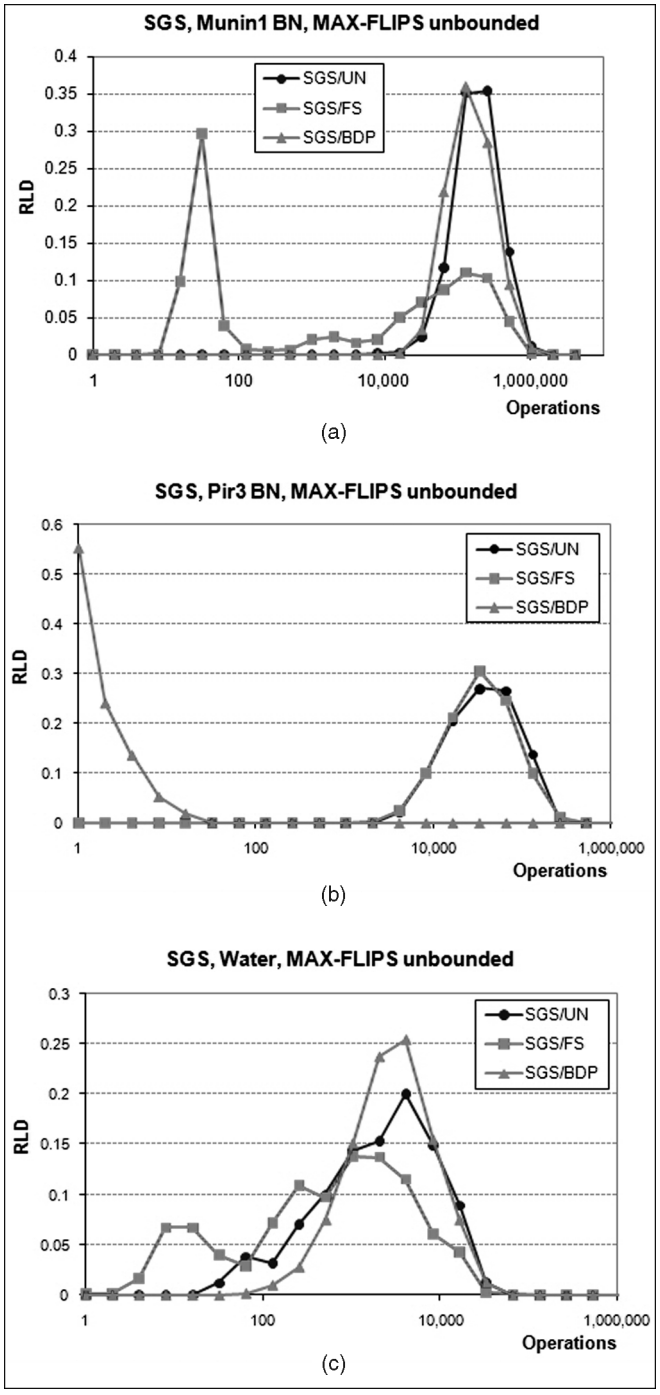

Fig. 4. Varying the initialization algorithms for three BNs. For each BN, three different variants of SGS are tested, namely SGS/UN, SGS/FS, and SGS/DP. An empirical run length distribution is displayed for each combination. Each BN has a best initialization algorithm, namely (a) SGS/FS for Munin1, (b) SGS/BDP for Pir3, and (c) SGS/FS for Water.

algorithms. We compare the performance of SGS and clique tree clustering, a state-of-the-art exact method, in Section 5.5.

5.1 Experimental Methodology

The main purpose of the empirical component of this research is scientific experimentation rather than competitive experimentation [41].[4] While we have highlighted the importance of initialization and introduced a novel Viterbi-based initialization approach, the purpose of our experiments is not to show that this approach is faster than existing algorithms on all problem instances (competitive experimentation). Rather, we aim to complement the discussion earlier in this paper (scientific experimentation), and, in particular, provide further details regarding the effect of using different initialization algorithms and varying MAX-FLIPS.

There is evidence that a problem instance that is hard for one SLS algorithms is also hard for other SLS algorithms [24], therefore we here investigate one SLS system in depth instead of performing superficial experiments with a large number of SLS systems. The SLS system used for experimentation was an implementation of SGS [14], [15]. Initialization algorithms, discussed in Section 3, were used in $\mathbb{I}$, namely UN—based on uniform initialization; FS—based on forward simulation; BDP—based on GRAPHBDP; and FDP—based on GRAPHFDP. Instance-specific search portfolios $\mathbb{S}$ were used in the experiments. These $\mathbb{S}$ always contained noisy search algorithms with nonzero selection probabilities (see Definition 5.5), thus SGS could always escape local but nonglobal maxima even for MAX-FLIPS $= \infty$. Input parameters $\ell = \Pr(\boldsymbol{x}^*)$, $\boldsymbol{e} = \{\}$, and MAX-TRIES $= \infty$ were given to SGS (except in Section 5.5 where MAX-TRIES $< \infty$ was used). The benefit of MAX-TRIES $= \infty$, often called the Las Vegas approach [42], [43], [12], is that one does not confound empirical results with the question of when to terminate. Using SGS, at least 1,000 repetitions (with different random seeds) were performed for each experiment reported here. In some cases, up to 10,000 or 100,000 repetitions were run.

We investigate the performance of SGS on BNs from applications; see elsewhere for synthetic BN results [17], [18]. The 10 distinct application BNs considered, most of which are taken from Friedman's Bayesian Network Repository, are shown in Table 5. (At the time of this writing, the location of the Bayesian Network Repository is at http://www.cs.huji.ac.il/labs/compbio/Repository/.) We discuss in greatest detail the Munin1, Pir3, and Water BNs. The Munin BNs are from the medical field of electromyography. The Pir BNs perform information filtering for the purpose of improving battlefield situation awareness [44]. The Water BN models the biological processes of water purification.

An Intel Pentium 4 2 GHz CPU with 1 GB of RAM, running Windows XP, was used in the experiments.

5.2 Varying the Initialization Algorithm

The purpose of the initialization experiments was to study in detail the effect, on empirical RLDs $\hat{Z}(a, \infty)$, or $\hat{Z}(a)$, and $\hat{Y}(a, \infty) = \hat{Z}(a, \infty) - 1$, or $\hat{Y}(a)$, of using different initialization algorithms. Three variants of SGS/a, namely SGS/UN, SGS/FS, and SGS/BDP, were used [14], [15]. Each SGS variant was tested using three BNs—Munin1, Pir3, and Water—giving a total of nine combinations. For each combination, 1,000 repetitions were executed. Following standard SLS methodology, no restarts were done at this stage [26], [12].

Results of these experiments are presented in Fig. 4 and Tables 1 and 2. Tables 1 and 2 were created using the WEKA implementation [45] of the expectation maximization (EM) algorithm. WEKA was used in two different modes, namely 1) to compute the optimal number of finite mixture

4. Hooker, in fact, uses the term "testing" rather than "experimentation" [41]; however, we here prefer the latter term.

TABLE 1
Mixture Models Computed Using the EM Algorithm over SGS Run Length Data, Using Three Different Initialization Algorithms and Three Different BNs

BN	Init.	Ln	κ^*	$\hat{\pi}_1$	$\hat{\mu}_1$	$\hat{\sigma}_1$	$\hat{\pi}_2$	$\hat{\mu}_2$	$\hat{\sigma}_2$	LL^*	AIC^*
Munin1	UN	n	5	0.22	58,206.54	21,709.53	0.31	113,821.39	25,923.72	-12.81	53.61
Munin1	UN	y	2	0.31	11.45	0.80	0.69	11.93	0.57	-1.03	12.06
Munin1	FS	n	8	0.43	21.84	7.00	0.07	979.74	766.79	-9.15	64.30
Munin1	FS	y	5	0.43	3.04	0.32	0.03	5.14	1.43	-1.88	31.75
Munin1	BDP	n	5	0.26	51,803.76	16,944.68	0.36	96,441.14	27,014.92	-12.62	53.25
Munin1	BDP	y	1	1.00	11.58	0.69				-1.05	6.09
Pir3	UN	n	5	0.21	8,311.19	3,047.46	0.35	20,489.07	7,239.97	-11.36	50.72
Pir3	UN	y	3	0.36	9.25	0.57	0.37	10.25	0.39	-1.24	18.49
Pir3	FS	n	6	0.22	7,830.90	2,911.51	0.25	16,735.10	4,370.79	-11.29	56.58
Pir3	FS	y	2	0.54	9.53	0.71	0.46	10.65	0.58	-1.26	12.51
Pir3	BDP	n	2	0.74	1.29	0.46	0.26	4.16	2.72	-1.48	12.96
Pir3	BDP	y	1	1.00	0.47	0.61				-0.92	5.85
Water	UN	n	8	0.18	169.94	110.59	0.26	748.22	332.40	-9.00	64.00
Water	UN	y	5	0.06	3.79	0.40	0.23	5.69	0.66	-1.77	31.54
Water	FS	n	4	0.20	14.10	11.58	0.36	320.87	232.10	-8.08	38.17
Water	FS	y	5	0.20	2.28	0.66	0.29	5.15	0.73	-2.08	32.15
Water	BDP	n	8	0.17	441.78	217.74	0.31	1,252.41	411.89	-9.04	64.09
Water	BDP	y	2	0.44	7.00	1.08	0.56	8.01	0.84	-1.48	12.96

Cross-validation results for both raw data and $\ln$*-transformed data are displayed (Ln column), as are the number of mixture components (*κ^**), the statistics for the one or two leftmost mixture components, as well as the log-likelihoods (*LL^**) and Akaike information criterion (*AIC^**) values.*

components κ^* by means of cross validation (see Table 1) and 2) to compute the mixture parameters for a fixed number of mixture components (see Table 2). Since normality is assumed by this variant of EM, the $\ln$-transformation performed as indicated in Table 1 means that log-normality is implied. These experiments are complementary to previous experiments by Hoos, where finite mixtures were used to model SLS performance on SAT instances from the hard region [26]. In particular, Hoos' experiments focused on synthetic SAT instances rather than application BNs and used at most two mixture components in experiments [26].

For the Munin1 BN, SGS/FS is the best performer. On average, SGS/FS, uses only 32 percent of the flips required for SGS/UN in order to find an MPE in these experiments. By investigating the RLDs in Fig. 4, it becomes clear that SGS/FS has a significant portion of much shorter searches than SGS/UN and SGS/BDP. This is reflected in Table 1, where the leftmost mixture component has for the raw data $\hat{\pi}_1 = 0.43$, $\hat{\mu}_1 = 21.84$, and $\hat{\sigma}_1 = 21.84$. For SGS/UN and SGS/BDP, there is no similar effect of short search lengths. The 25th percentile is 87,215 flips for SGS/UN and 65,215 flips for SGS/BDP. In Table 2, there is for SGS/FS and Munin1 a prominent drop in the log-likelihood from $\kappa = 1$ to $\kappa = 2$, indicating that much RLD probability mass is quite accurately accounted for when there are two components in the mixture.

For the Pir3 BN, SGS/BDP is extremely strong. In fact, in over 50 percent of the cases an MPE is found as a result of initialization (zero flips). Table 1 reflects this strong performance, with $\hat{\pi}_1 = 0.74$ and $\hat{\mu}_1 = 0.46$. In contrast, the averages for Pir3 for SGS/UN and SGS/FS are $E(\hat{Y}(a,\infty)) = 34,469$ flips and $E(\hat{Y}(a,\infty)) = 32,249$ flips, respectively. Pir3's backward tree-like structure appears to explain the strong results for SGS/BDP.

For the Water BN, SGS/FS is on average approximately twice as fast as SGS/UN and SGS/BDP. For SGS/FS, we see in Fig. 4 a rapid increase in the empirical RLD until approximately 10 operations. In fact, for Water the first percentile is three flips and the fifth percentile is six flips. Table 1 and Table 2 reflect similar trends for SGS/FS compared to its alternatives.

These experiments show that initialization algorithms can have a major positive impact on the SGS inference time. For each of these BNs, there is one initialization algorithm that substantially outperforms the traditional approach of initializing uniformly at random, here implemented in SGS/UN. For some initialization algorithms, a nontrivial percentage of searches turned out to be relatively short as shown in Fig. 4. For Pir3, SGS/BDP has a 99th percentile of nine flips; for Munin1, SGS/FS has a 25th percentile of 21 flips; and for Water, SGS/FS has a 25th percentile of 91 flips.

These experiments exhibit clear evidence of mixture distribution behavior, aligning well with the discussion in Section 4.2. We see in Table 1 that the optimized number of mixture components, κ^*, ranges from 1 to 8. In particular, in Table 1, the use of two or more mixture components was found to be optimal in most cases: $\kappa^* \geq 2$ in 16 out of 18 rows in the table.

TABLE 2
Mixture Models Generated Using the EM Algorithm over SGS Run Length Data, Using Three Different Initialization Algorithms, for the BNs Munin1, Pir3, and Water

BN	Init.	Ln	$\kappa = 1$	$\kappa = 2$	$\kappa = 3$	κ^*	LL^*
Munin1	UN	n	-13.02	-12.85	-12.82	5	-12.81
Munin1	FS	n	-12.82	-9.50	-9.31	8	-9.15
Munin1	BDP	n	-12.94	-12.70	-12.64	5	-12.62
Pir3	UN	n	-11.65	-11.44	-11.38	5	-11.36
Pir3	FS	n	-11.65	-11.40	-11.33	6	-11.29
Pir3	BDP	n	-2.06	-1.48	-1.98	2	-1.48
Water	UN	n	-9.71	-9.28	-9.11	8	-9.00
Water	FS	n	-9.41	-8.60	-8.30	4	-8.08
Water	BDP	n	-9.60	-9.20	-9.11	8	-9.04

Log-likelihoods (LLs) for $\kappa = 1, 2, 3$ *mixture components are shown. The two right-most columns show the number of mixture components* κ^* *and log-likelihood* LL^* *computed by cross validation.*

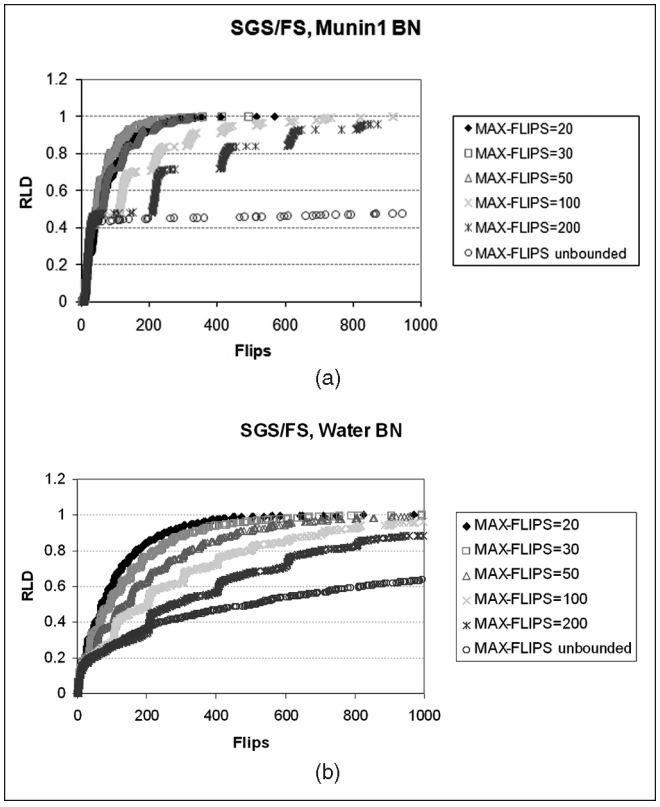

Fig. 5. Empirical RLDs for SGS when varying the restart parameter from $\text{MAX-FLIPS} = 20$ to MAX-FLIPS unbounded.

5.3 Varying the Restart Parameter

The purpose of the restart experiments was to investigate the effect, on empirical RLDs $\hat{Y}(a, m)$, of varying the SLS restart parameter MAX-FLIPS. The MAX-FLIP parameter was set to $\text{MAX-FLIPS} = 20$, $\text{MAX-FLIPS} = 30$, $\text{MAX-FLIPS} = 50$, $\text{MAX-FLIPS} = 100$, and $\text{MAX-FLIPS} = 200$, and for each condition, the MPE was computed 1,000 times using SGS while gathering statistics for RLD estimates $\hat{Y}(a, m)$.

In Fig. 5, the $\hat{Y}(a, m)$ results for SGS/FS using the Munin1 BN are shown. Varying MAX-FLIPS has a major impact, since for the unbounded case $\text{MAX-FLIPS} = \infty$ the RLD is almost horizontal for RLD-values slightly greater than 0.4. For the larger values of MAX-FLIPS, except for the unbounded MAX-FLIPS case, we observe a stair-case-shaped RLD, where steps get progressively smaller as the number of flips increases. Clearly, the reason for this pattern is the substantial probability mass in the left tail. This is also shown in Table 1, where the leftmost mixture component has for SGS/FS $\hat{\pi}_1 = 0.43$, $\hat{\mu}_1 = 21.84$, and $\hat{\sigma}_1 = 7.00$.

Results for $\hat{Y}(a, m)$ for SGS/FS using the Water BN are shown in Fig. 5. While the effect here is not as dramatic as for Munin1, $\text{MAX-FLIPS} = 20$ gives, in general, visibly better performance than the other MAX-FLIPS settings. Again, this behavior reflects Table 1, where the leftmost mixture component has for SGS/FS $\hat{\pi}_1 = 0.20$, $\hat{\mu}_1 = 14.10$, and $\hat{\sigma}_1 = 11.58$.

For Pir3, varying the restart parameter had a minimal effect on $\hat{Y}(a, m)$, as one might expect given the similarity of the RLD to an exponential distribution (see Fig. 4), and to save space we do not report details.

In summary, we have seen that varying the MAX-FLIPS parameter can have a substantial impact on SGS run lengths for certain BNs. This leads to the following question. What are the optimal or near-optimal values of MAX-FLIPS for such BNs? This issue is what we turn to in the experiments of the next section.

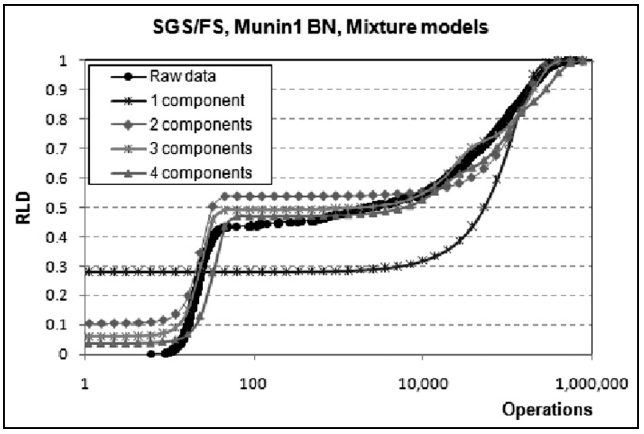

Fig. 6. Empirical run length distribution for SGS/FS for Munin1 along with four mixture models, with varying number of mixture components.

5.4 Optimization of Initialization and Restart

The purpose of the optimization experiments was to empirically find optimal or near-optimal values $\hat{m}_f^*$ of MAX-FLIPS, thus minimizing $E(\hat{Y}(a, m))$. The speedup that can be obtained by optimizing both initialization and restart was of interest. Compared to traditional optimization, we note that SLS restart parameter optimization is complicated by the noise in the objective function caused by SLS randomization.

Based on pilot studies reported in Section 5.3, we investigated optimization of SGS for Munin1, Pir3, and Water in detail. Using SGS/FS and Munin1, MAX-FLIPS was varied from $\text{MAX-FLIPS} = 20$ to $\text{MAX-FLIPS} = 40$. For each setting, 10,000 Las Vegas run length experiments were performed. Both the number of flips and computation times were recorded, and then the sample averages for number of flips $E(\hat{Y}(a, m))$ and execution times $E(\hat{C}(a, m))$ were computed.

Results from these Munin1 experiments are shown in Fig. 7. Fig. 7 contains, in addition to the data points, fourth order polynomial regression results. Similar results are reported for Water BN. Given the approximate polynomials above, one can find approximate optima by setting the derivative to zero, $f'(x) = 0$, and solving for x, giving results that are very similar to those obtained using just run length. Further insight and confirmation is provided by Fig. 6 and Table 3, which show raw data along with normal mixture models. Using the probabilistic approach discussed in Section 4, in particular (14), these mixture models were used to obtain estimates $\hat{m}_o^*$(FS) as summarized in Table 3.

A summary of the results for the different BNs is provided in Table 4. The baseline SGS system is using uniform initialization and no restart. Optimized SGS is using the best initialization algorithm (for that particular BN) and an optimized value for MAX-FLIPS (MF). For the Pir3 BN, the mean of the baseline approach SGS/UN was 34,469 flips while optimized SGS/BDP used 0.8621 flips. In terms of wall-clock time, optimized SGS/BDP was 6,629 times faster than unoptimized SGS/UN. There are similar, but less dramatic, speedups for the Munin1 and Water BNs.

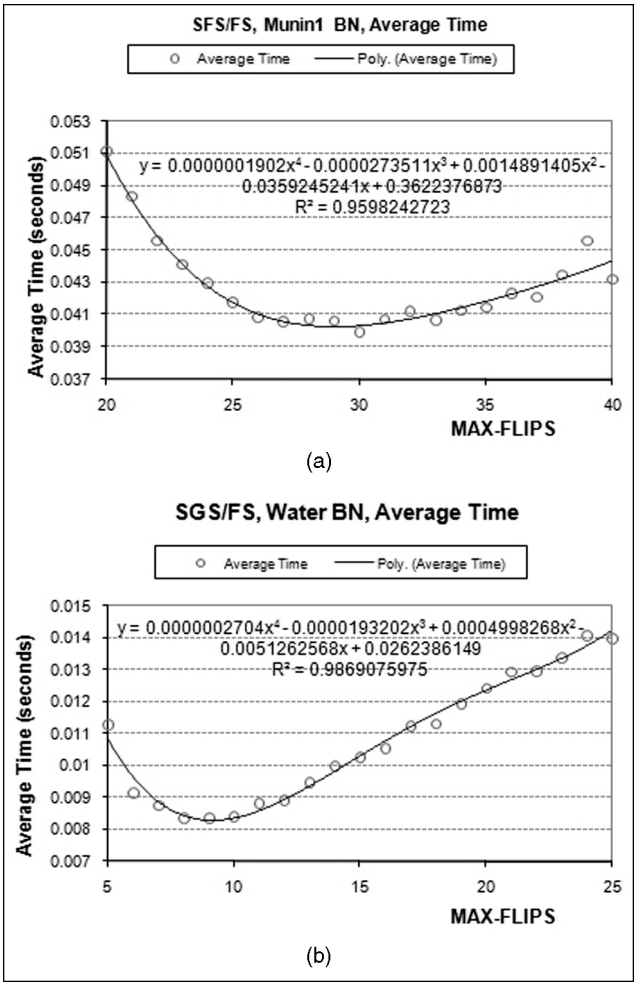

Fig. 7. Varying the MAX-FLIPS restart parameter for SGS/FS on the BNs Munin1 (a) and Water (b). Average computation time is displayed for each BN. Approximations using fourth order polynomials, based on the experimental data points, are also shown.

Overall, the speedups obtained by carefully optimizing initialization and restart are quite dramatic for these application BNs. Initialization is more important than restart, in the sense that strong performance can be obtained for a particular initialization algorithm a for SGS/a even without optimizing MAX-FLIPS.

5.5 Comparison to Clique Tree Clustering

The purpose of these experiments was to compare SGS with clique tree clustering [6], [7], [33], [8] as implemented in the state-of-the-art system HUGIN. Clique tree clustering is an exact algorithm that consists of two phases, the clustering phase and the propagation phase. The clique tree propagation phase operates on a clique tree, created from a BN by the clustering phase. Clique tree size upper bounds treewidth, a fundamental graph-theoretic parameter [46] of a BN.

TABLE 4
Optimizing Performance of SGS for Three BN Instances

	Baseline SGS		Optimized SGS		Speedup ratio	
BN	Flips	MF	Flips	MF	Flips	Time
Munin1	162,373	∞	61.09	30	2,680:1	2,656:1
Pir3	34,469	∞	0.8621	6	39,982:1	6,629:1
Water	3,121	∞	70.59	9	44.2:1	38.34:1

TABLE 5
Comparison of Performance of SGS and Clique Tree Propagation on Different BN Instances

BN		SGS			CTP	
Name	m/n	Init.	MF	Time (s)	Size (k)	Time (s)
Mildew	1.31	FDP	200	0.257	9,566	3.88
Munin1	1.49	FS	30	0.0254	384,621	1824
Munin2	1.24	FS	113	1.98	4,862	2.75
Munin3	1.26	-	-	-	3,113	0.744
Munin4	1.34	-	-	-	14,340	3.19
Pigs	1.34	FDP	550	3.41	828	15.7
Pir1	1.02	BDP	2	0.0205	11	0.0197
Pir2	1.00	BDP	8	0.0100	8	0.0367
Pir3	1.17	BDP	6	0.00216	5	0.00096
Water	2.06	FS	9	0.00837	3,657	0.772

Table 5 presents experimental results for 10 distinct application BNs for both SGS and clique tree propagation. In these experiments, where the SGS system was optimized using our technique discussed earlier in the paper, we only account for online runtime, not preprocessing time. For HUGIN, preprocessing amounts to compilation of a BN into a clique tree. For SGS, preprocessing amounts to optimizing the parameters controlling search. The results in Table 5 have for HUGIN been averaged over 50 runs, for SGS over 1,000 runs.

Total clique tree size and inference time, as computed by clique tree clustering, is an indication of BN hardness. Among these BNs, Munin1 has the largest total clique tree size and the slowest execution time. SGS clearly outperforms CTP on Mildew, Munin1, Pigs, Pir2, and Water. CTP is clearly superior to SGS for Munin3 and Munin4, as SGS did not terminate within the allocated number of MAX-TRIES for these BNs.

We now compare the performance of the novel initialization algorithms proposed in this paper, BDP and FDP, with FS. A key difference is that BDP and FDP are structure-based, while FS is CPT-based. As a consequence, BDP and FDP can be expected to perform very well for BNs that are tree-structured or close to tree-structured, and not so well otherwise. FS, on the other hand, does not rely on structure. The experimental results confirm these qualitative statements: Table 5 illustrates how BDP and FDP are the best performers, relative to FS, for BNs with substantial tree

TABLE 3
Mixture Models, with $\kappa = 1$ to $\kappa = 4$ Components, Generated Using the Expectation Maximization Algorithm

κ	$\hat{\pi}_1$	$\hat{\mu}_1$	$\hat{\sigma}_1$	$\hat{\pi}_2$	$\hat{\mu}_2$	$\hat{\sigma}_2$	$\hat{\pi}_3$	$\hat{\mu}_3$	$\hat{\sigma}_3$	$\hat{\pi}_4$	$\hat{\mu}_4$	$\hat{\sigma}_4$	LL	$\hat{m}_o^*$(FS)
1	1.0	52,081	89,780										-12.82	-
2	0.433	21.94	7.166	0.567	91,795	102,816							-9.50	32
3	0.432	21.92	7.129	0.231	15,029	13,620	0.337	144,449	104,328				-9.31	32
4	0.432	21.91	7.108	0.157	7,384	7,025	0.211	55,954	31,719	0.200	19,5384	10,5862	-9.25	32

structure, reflected in a small ratio m/n for a BN $\beta = (\boldsymbol{X}, \boldsymbol{E}, \boldsymbol{P})$, where $n = |\boldsymbol{X}|$ and $m = |\boldsymbol{E}|$.

In summary, these experiments clearly show that our Stochastic Greedy Search algorithm—specifically SGS/FS, SGS/BDP, and SGS/FDP—performs very well on nontrivial BNs. In particular, these experiments show competitive performance on BNs that are not trivial for clique tree clustering, a state-of-the-art algorithm for BN inference.

6 Discussion and Future Work

Stochastic local search algorithms provide, for certain classes of applications and problem instances, an appealing trade-off of accuracy for speed and memory requirements. Specifically, SLS algorithms require little memory and are very fast for certain problem instances, but are incomplete and may produce suboptimal results. By focusing on computing most probable explanations in Bayesian networks, we have, in this paper, presented algorithmic, analytical, and experimental results regarding SLS initialization (where to start search?) and the SLS restart parameter (when to restart search?). In particular, we have discussed Viterbi-based initialization algorithms, an analytical framework for SLS analysis, an analysis of SGS specifically, and finally SGS's competitive performance relative to clique tree clustering. By carefully and jointly optimizing the initialization algorithm and the restart parameter MAX-FLIPS for SGS, we improved for application BNs the search performance by several orders of magnitude compared to initialization uniformly at random with no restart (or $\text{MAX-FLIPS} = \infty$).

We now consider, for SGS, the optimization of the initialization algorithm a and the MAX-FLIPS parameter. This optimization—which is reflected in the progression of Sections 5.2, 5.3, and 5.4—is a heuristic preprocessing step. First, as discussed in Section 5.2, one typically optimizes the selection probabilities in the initialization portfolio $\mathbb{I}$. This optimization is partly informed by the structure of the BN, such that the probability of picking FDP (say) from the initialization portfolio is set higher for a BN that is close to having a forward tree structure. The probabilities are then gradually adjusted, as one sees the impact of the different initialization algorithms on the progress of the stochastic search process, until one is left with a homogenous initialization portfolio SGS/a. After a has been identified, the emphasis typically shifts to the optimization of the MAX-FLIPS parameter, utilizing the mixture distribution properties of the run length distributions identified in this paper. Empirical aspects of this optimization are investigated in Sections 5.3 and 5.4.

Our results appear to create new research opportunities in the area of adaptive and hybrid SLS algorithms. One opportunity concerns the development of new initialization algorithms. For example, it would be of interest to investigate more fine-grained hybrid algorithms, where different initialization algorithms may be applied to create different subexplanations. Another question is whether SGS performance can be further improved by automatically optimizing the input parameters. Finally, additional work on finite mixture models appears useful. Most current approaches assume, as we have done here, homogenous mixtures. We speculate that heterogenous mixture models, where one distribution function F_1 is (say) normal while another distribution function F_2 is (say) exponential, could provide improved fits for certain run length distributions, and would be useful in other contexts as well.

Acknowledgments

This material is based, in part, upon work by Ole J. Mengshoel supported by NASA award NCC2-1426 as well as the US National Science Foundation (NSF) grants CCF-0937044 and ECCS-0931978. Ole J. Mengshoel and David C. Wilkins gratefully acknowledge support in part by ONR grant N00014-95-1-0749, ARL grant DAAL01-96-2-0003, and NRL grant N00014-97-C-2061. Dan Roth gratefully acknowledges the support of NSF grants IIS-9801638 and SBR-987345. David Fried and Song Han are acknowledged for their codevelopment of the Raven software, used in experimental work reported here. Comments from the anonymous reviewers, which helped improve the paper, are also acknowledged.

References

[1] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[2] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Information Theory,* vol. 47, no. 2, pp. 498-519, Feb. 2001.

[3] M. Wainwright, T. Jaakkola, and A. Willsky, "MAP Estimation via Agreement on (Hyper)Trees: Message-Passing and Linear Programming Approaches," *IEEE Trans. Information Theory,* vol. 51, no. 11, pp. 3697-3717, Nov. 2002.

[4] M.J. Wainwright, T.S. Jaakkola, and A.S. Willsky, "Tree-Based Reparameterization Framework for Analysis of Sum-Product and Related Algorithms," *IEEE Trans. Information Theory,* vol. 49, no. 5, pp. 1120-1146, May 2003.

[5] A. Darwiche, "A Differential Approach to Inference in Bayesian Networks," *J. ACM,* vol. 50, no. 3, pp. 280-305, 2003.

[6] S. Lauritzen and D.J. Spiegelhalter, "Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems (with Discussion)," *J. Royal Statistical Soc. Series B,* vol. 50, no. 2, pp. 157-224, 1988.

[7] S.K. Andersen, K.G. Olesen, F.V. Jensen, and F. Jensen, "HUGIN—A Shell for Building Bayesian Belief Universes for Expert Systems," *Proc. 11th Int'l Joint Conf. Artificial Intelligence,* vol. 2, pp. 1080-1085, Aug. 1989.

[8] C. Huang and A. Darwiche, "Inference in Belief Networks: A Procedural Guide," *Int'l J. Approximate Reasoning,* vol. 15, pp. 225-263, 1996.

[9] B. Selman, H. Levesque, and D. Mitchell, "A New Method for Solving Hard Satisfiability Problems," *Proc. 10th Nat'l Conf. Artificial Intelligence (AAAI '92),* pp. 440-446, 1992.

[10] B. Selman, H.A. Kautz, and B. Cohen, "Noise Strategies for Improving Local Search," *Proc. 12th Nat'l Conf. Artificial Intelligence (AAAI '94),* pp. 337-343, 1994.

[11] P.W. Gu, J. Purdom, J. Franco, and B.W. Wah, "Algorithms for the Satisfiability SAT Problem: A Survey," *Satisfiability Problem: Theory and Applications,* pp. 19-152. Am. Math. Soc., 1997.

[12] H.H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann, 2005.

[13] K. Kask and R. Dechter, "Stochastic Local Search for Bayesian Networks," *Proc. Seventh Int'l Workshop Artificial Intelligence and Statistics,* Jan. 1999.

[14] O.J. Mengshoel, "Efficient Bayesian Network Inference: Genetic Algorithms, Stochastic Local Search, and Abstraction," PhD dissertation, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Apr. 1999.

[15] O.J. Mengshoel, D. Roth, and D.C. Wilkins, "Stochastic Greedy Search: Computing the Most Probable Explanation in Bayesian Networks," Technical Report UIUCDCS-R-2000-2150, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Feb. 2000.

[16] F. Hutter, H.H. Hoos, and T. Stützle, "Efficient Stochastic Local Search for MPE Solving," *Proc. 19th Int'l Joint Conf. Artificial Intelligence (IJCAI '05),* pp. 169-174, 2005.

[17] O.J. Mengshoel, "Understanding the Role of Noise in Stochastic Local Search: Analysis and Experiments," *Artificial Intelligence,* vol. 172, nos. 8-9, pp. 955-990, 2008.

[18] O.J. Mengshoel, D. Roth, and D.C. Wilkins, "Portfolios in Stochastic Local Search: Efficiently Computing Most Probable Explanations in Bayesian Networks," *J. Automated Reasoning,* published online 14 Apr. 2010.

[19] J.D. Park and A. Darwiche, "Complexity Results and Approximation Strategies for MAP Explanations," *J. Artificial Intelligence Research,* vol. 21, pp. 101-133, 2004.

[20] B. Selman and H. Kautz, "Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI '93),* pp. 290-295, 1993.

[21] R. Lin, A. Galper, and R. Shachter, "Abductive Inference Using Probabilistic Networks: Randomized Search Techniques," Technical Report KSL-90-73, Knowledge Systems Laboratory, Nov. 1990.

[22] L.R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE,* vol. 77, no. 2, pp. 257-286, Feb. 1989.

[23] H.H. Hoos and T. Stützle, "Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT," *Artificial Intelligence,* vol. 112, nos. 1-2, pp. 213-232, 1999.

[24] H.H. Hoos and T. Stützle, "Local Search Algorithms for SAT: An Empirical Evaluation," *J. Automated Reasoning,* vol. 24, no. 4, pp. 421-481, citeseer.ist.psu.edu/hoos99local.html, 2000.

[25] D. Mitchell, B. Selman, and H.J. Levesque, "Hard and Easy Distributions of SAT Problems," *Proc. 10th Nat'l Conf. Artificial Intelligence (AAAI '92),* pp. 459-465, 1992.

[26] H.H. Hoos, "A Mixture-Model for the Behaviour of SLS Algorithms for SAT," *Proc. 18th Nat'l Conf. Artificial Intelligence (AAAI '02),* pp. 661-667, 2002.

[27] I.P. Gent and T. Walsh, "Easy Problems are Sometimes Hard," *Artificial Intelligence,* vol. 70, nos. 1-2, pp. 335-345, 1994.

[28] A.J. Parkes and J.P. Walser, "Tuning Local Search for Satisfiability Testing," *Proc. 13th Nat'l Conf. Artificial Intelligence (AAAI '96),* pp. 356-362, citeseer.ist.psu.edu/parkes96tuning.html, 1996.

[29] C.P. Gomes, B. Selman, and H. Kautz, "Boosting Combinatorial Search through Randomization," *Proc. 15th Nat'l Conf. Artificial Intelligence (AAAI '98),* pp. 431-437, 1998.

[30] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and D. Chickering, "A Bayesian Approach to Tackling Hard Computational Problems," *Proc. 17th Ann. Conf. Uncertainty in Artificial Intelligence (UAI '01),* pp. 235-244, 2001.

[31] Y. Ruan, E. Horvitz, and H. Kautz, "Restart Policies with Dependence among Runs: A Dynamic Programming Approach," *Proc. Eighth Int'l Conf. Principles and Practice of Constraint Programming,* pp. 573-586, 2002.

[32] Y. Ruan, E. Horvitz, and H. Kautz, "Hardness-Aware Restart Policies," *Proc. 18th Int'l Joint Conf. Artificial Intelligence (IJCAI '03) Workshop Stochastic Search Algorithms,* 2003.

[33] A.P. Dawid, "Applications of a General Propagation Algorithm for Probabilistic Expert Systems," *Statistics and Computing,* vol. 2, pp. 25-36, 1992.

[34] E. Shimony, "Finding MAPs for Belief Networks is NP-Hard," *Artificial Intelligence,* vol. 68, pp. 399-410, 1994.

[35] A.M. Abdelbar and S.M. Hedetnieme, "Approximating MAPs for Belief Networks is NP-Hard and Other Theorems," *Artificial Intelligence,* vol. 102, pp. 21-38, 1998.

[36] A. Viterbi, "Error Bounds for Convolutional Codes and An Asymptotically Optimal Decoding Algorithm," *IEEE Trans. Information Theory,* vol. 13, no. 2, pp. 260-269, Apr. 1967.

[37] M. Henrion, "Propagating Uncertainty in Bayesian Networks by Probabilistic Logic Sampling," *Uncertainty in Artificial Intelligence,* vol. 2, pp. 149-163, Elsevier, 1988.

[38] R. Dechter and I. Rish, "Mini-Buckets: A General Scheme for Bounded Inference," *J. ACM,* vol. 50, no. 2, pp. 107-153, 2003.

[39] R. Dechter, "Bucket Elimination: A Unifying Framework for Reasoning," *Artificial Intelligence,* vol. 113, nos. 1-2, pp. 41-85, citeseer.nj.nec.com/article/dechter99bucket.html, 1999.

[40] D. Schuurmans and F. Southey, "Local Search Characteristics of Incomplete SAT Procedures," *Artificial Intelligence,* vol. 132, no. 2, pp. 121-150, citeseer.ist.psu.edu/article/schuurmans00local.html, 2001.

[41] J. Hooker, "Testing Heuristics: We have It All Wrong," *J. Heuristics,* vol. 1, pp. 33-42, 1996.

[42] R. Motwani and P. Raghavan, *Randomized Algorithms.* Cambridge Univ. Press, 1995.

[43] B.A. Huberman, R.M. Lukose, and T. Hogg, "An Economics Approach to Hard Computational Problems," *Science,* vol. 275, no. 3, pp. 51-54, 1997.

[44] P. Jones, C. Hayes, D. Wilkins, R. Bargar, J. Sniezek, P. Asaro, O.J. Mengshoel, D. Kessler, M. Lucenti, I. Choi, N. Tu, and J. Schlabach, "CoRAVEN: Modeling and Design of a Multimedia Intelligent Infrastructure for Collaborative Intelligence Analysis," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics,* pp. 914-919, Oct. 1998.

[45] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques,* second ed., Morgan Kaufmann, 2005.

[46] H.L. Bodlaender, "A Tourist Guide through Treewidth," *Acta Cybernetica,* vol. 11, pp. 1-21, citeseer.nj.nec.com/bodlaender93tourist.html, 1993.

Ole J. Mengshoel received the undergraduate degree in computer science from the Norwegian Institute of Technology (now, NTNU) and the PhD degree in computer science from the University of Illinois, Urbana-Champaign. He is currently a senior systems scientist with Carnegie Mellon University (CMU), Silicon Valley, and also is affiliated with the NASA Ames Research Center, Moffett Field, California. Prior to joining CMU, he was a senior scientist and research area lead at USRA/RIACS; a research scientist in the Decision Sciences Group at Rockwell Scientific; and in the Knowledge-Based Systems Group at SINTEF, Norway. At NASA, he has a leadership role in the Diagnostics and Prognostics Group in the Intelligent Systems Division, where his current research focuses on reasoning, machine learning, diagnosis, prognosis, and decision support under uncertainty—often using Bayesian networks—with aerospace applications of interest to NASA. He is a member of the IEEE, the AAAI, and the ACM, and has numerous times served as reviewer and on Program Committees. He has published more than 50 articles and papers in journals, conferences, and workshops, and holds four US patents.

David C. Wilkins received the PhD degree from the University of Michigan in 1987. His PhD dissertation research was carried out in the Department of Computer Science at Stanford University between 1982 and 1987. His current affiliations at Stanford are with the Symbolic Systems Program, which focuses on a science of the mind, and the Stanford Center for Creativity in the Arts (SICa), where he serves on an Advisory committee. He has a senior research affiliate position at the Institute for the Study of Learning and Expertise (ISLE) in Palo Alto. Prior to returning to Stanford, he was on the faculty at the University of Illinois at Urbana-Champaign from 1988-2005, with faculty appointments in Computer Science, Psychology, Aviation Institute, and Beckman Institute. His research area within artificial intelligence and cognitive science is computational models of human learning, decision making, and expertise. His research specialty is interactive learning environments, especially apprenticeship learning systems for learning and teaching expert decision making, and his research projects typically involve faculty collaborators from psychology, computer science, or linguistics.

Dan Roth received the BA degree with Summa cum laude in mathematics from the Technion, Israel, and the PhD degree in computer science from Harvard University in 1995. He is a professor in the Department of Computer Science and the Beckman Institute at the University of Illinois at Urbana-Champaign. He is the director of the DHS funded center for Multimodal Information Access & Synthesis (MIAS) and has faculty positions also at the Statistics and Linguistics Departments and at the graduate School of Library and Information Science. He is a fellow of the AAAI, and has published broadly in machine learning, natural language processing, knowledge representation, and reasoning and learning theory. He has developed advanced machine-learning-based tools for natural language applications, including an award winning Semantic Parser. He has given keynote talks in major conferences, including the AAAI, the Conference of the American Association of Artificial Intelligence, EMNLP, the Conference on Empirical Methods in Natural Language Processing, and ECML & PKDD, the European Conference on Machine Learning and the Principles and Practice of Knowledge Discovery in Databases. He was the program chair of CoNLL'02 and of ACL'03, and is or has been on the editorial board of several journals in his research areas. He is currently an associate editor for the *Journal of Artificial Intelligence Research* and the *Machine Learning Journal*. He is a member of the IEEE.